Fachhochschule Köln
University of Applied Sciences Cologne
Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften
Studiengang allgemeine Informatik

Diplomarbeit
zur Erlangung
des Diplomgrades
Diplom-Informatiker (FH)
in der Fachrichtung Informatik

Entwicklung einer graphischen Simulation von LEGO® Mindstorms™ Robotern

Erstprüferin:	Prof. Dr. Heide Faeskorn-Woyke
Zweitprüferin:	Monika Müllerburg
vorgelegt am:	5.3.2004
von cand.:	Björn Flintrop
aus	St. Ägidiusstr. 59
	51147 Köln
Mat.-Nr.:	1102 1497

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AIS	Autonome Intelligente Systeme
J2SE	Java™ 2 Standard Edition
JDK	Java™ Development Kit
JRE	Java™ Runtime Environment
MindSim	Mindstorms™ Simulator
NQC	Not Quite C
ODE	Open Dynamics Engine
ODL	Open Dynamics Language
OpenGL	Open Graphics Library
RCX	Robotic Command Explorer
RIS	Robotics Invention System™
SDK	Software Development Kit

1 Einleitung

Die vorliegende Diplomarbeit entstand an der Fakultät für Informatik und Ingenieurwissenschaften der Fachhochschule Köln, Campus Gummersbach, in Zusammenarbeit mit dem Projekt *Roberta*[1] am Fraunhofer Institut für Autonome Intelligente Systeme (Fraunhofer AIS) in Sankt Augustin. Sie besteht aus dieser schriftlichen Ausarbeitung und einer CD-ROM, die den Quelltext des entwickelten Produktes, das für Windows™ Systeme kompilierte, ausführbare Produkt selbst und die mit *JavaDoc*[2] erstellte Quelltextdokumentation enthält.

1.1 Ziel und Aufbau der Arbeit

Ziel dieser Diplomarbeit ist die Entwicklung eines Softwareproduktes. Zu entwickeln ist eine graphische Simulation von LEGO® Mindstorms™ Robotern. Die Simulation soll die Möglichkeit bieten, ein virtuelles Robotergrundmodell mit verschiedenen Sensorausstattungen so zu programmieren, wie es auch bei einem echten Modell der Fall wäre. Das real zu erwartende Verhalten dieses Roboters bei Programmausführung soll auf dem Bildschirm in Echtzeit verfolgt werden können, wobei die Bedienung des virtuellen Roboters der des realen Robotermodells nachempfunden sein soll.

Im Projekt »Roberta« werden Roboterkurse entwickelt und erprobt, die durch geschulte Roberta-KursleiterInnen durchgeführt werden. In diesen Kursen werden LEGO® Mindstorms™ Roboter eingesetzt.

---

[1] Im Rahmen der Kurse des Projekts »*Roberta* - Mädchen erobern Roboter« werden hauptsächlich Mindstorms™ Roboter eingesetzt. Das Projekt wird in Kapitel 1.3 näher erläutert.

[2] *JavaDoc* ist der in dem Java™ Software Development Kit (SDK) enthaltene Dokumentationsgenerator. Er erzeugt anhand von Java-Quelltexten eine Dokumentation, in der die jeweiligen Klassen, Interfaces, Methoden, Konstruktoren und Datenelemente aufgeführt sind.

Für die KursleiterInnen ist es sehr wichtig, sich mit der Programmierung der Mindstorms™ Roboter vertraut machen zu können, auch wenn die Roboter physikalisch nicht vorhanden sind.

Nach dieser Einleitung folgt in Kapitel 2 zunächst die Betrachtung einiger Grundlagen zu Robotersystemen. In Kapitel 3 findet sich das Pflichtenheft, welches Anforderungen und Zielbestimmung festlegt. Die grundlegenden Entwurfsentscheidungen und Werkzeuge werden in Kapitel 4 behandelt. Danach erfolgt in Kapitel 5 und 6 die Analyse der Mindstorms™ Roboter und der zu entwickelnden Komponenten. Dazu werden auch Lösungs- bzw. Implementationsansätze erläutert. In Kapitel 5 werden das Betriebssystemkonzept des RCX und die Möglichkeiten einer Emulationsentwicklung dargestellt. In Kapitel 6 werden die technischen Eigenschaften eines Robotermodells näher beleuchtet und die zu entwickelnden Komponenten zur Berechnung und Darstellung dieses Roboters beschrieben. Es folgt in Kapitel 7 die Präsentation des fertigen Softwareproduktes mit Installationshinweisen und Benutzungsanleitung. Schließlich findet sich in Kapitel 8 ein Resümee über die vorliegende Arbeit mit einem Ausblick auf zukünftige Weiterentwicklungen.

1.2 LEGO® Mindstorms™ Roboter

Seit ihrer Markteinführung im Herbst 1998 erfreuen sich LEGO® Mindstorms™ Roboter sowohl bei der eigentlichen Zielgruppe, Kindern und Jugendlichen ab einem Alter von 12 Jahren, als auch bei Erwachsenen größter Beliebtheit. Die verschiedenen Baukästen bieten die Möglichkeit, die Steuerung und Programmierung von autonomen Robotersystemen auf spielerische Art und Weise zu erlernen, wobei im Zusammenhang mit dem Konstruieren der Robotermodelle auch grundlegende Mechanikkentnisse erworben werden. Weiterhin eignen sie sich auch vorzüglich für die Entwicklung umfangreicher Programme, deren Ausführung die Roboter vielfältige Aufgaben erfüllen lässt.

Die Benutzung der beim Kauf eines *Robotics Invention System*™[3] (RIS) Baukastens von LEGO® mitgelieferten Programmierumgebung ist sehr schnell zu erlernen. Die rein graphisch basierte Erstellung von Programmen mit dieser Software ist vor allem für Anfänger intuitiv. Für fortgeschrittene AnwenderInnen, vor allem solche mit Programmierkenntnissen in textbasierenden Sprachen, stehen mehrere kostenfreie *Third Party Produkte*[4] zur Verfügung. Mit diesen ist eine Programmierung der Roboter mit gängigen Hochsprachen wie C, C++ oder Java möglich.

Nachdem der große Verkaufserfolg zunächst im Bereich der HeimanwenderInnen eingesetzt hatte, verbreiteten sich die Baukästen auch im Bereich von Ausbildung, Schulung und Forschung in zunehmendem Maße, sodass Mindstorms™ Roboter mittlerweile an vielen Bildungseinrichtungen, vor allem technischen Hochschulen und Fachhochschulen, zu finden sind und dort für die Lehre eingesetzt werden.

Neben der großen Bandbreite der Nutzungsmöglichkeiten in den genannten Bereichen spricht auch das gute Preis-Leistungsverhältnis für den Einsatz der Mindstorms™ Baukästen. Ein RIS Baukasten kostet im Spielwarenhandel ca. 250 Euro. Er enthält unter anderem das Kernstück jedes RIS Mindstorms™ Roboters: den »Robotic Command Explorer« (RCX). Dieser Baustein besteht aus einem Mikroprozessorsystem mit Infrarotschnittschnelle und jeweils drei Ein- und Ausgängen. Des Weiteren sind neben der Programmierumgebung noch zwei Motoren, zwei Berührungssensoren, ein Lichtsensor

---

[3] Innerhalb der Mindstorms™ Produktreihe sind die mit *Robotics Invention System*™ bezeichneten Baukästen die erfolgreichsten. Sie bieten im LEGO® Segment für Privatkunden (Spielwarenhandel, etc.) auch die meisten Möglichkeiten bezüglich der Freiheit in Konstruktion und Programmierung.

[4] *Third Party Produkte:* Produkte, die in Zusammenhang mit einem proprietären System (hier: LEGO® Mindstorms™) eingesetzt, jedoch nicht vom Markeneigner selbst entwickelt und vertrieben werden.

und insgesamt über 700 weitere Teile zum Bau der Robotermodelle enthalten.

Über die aktuellen Gesamtverkaufszahlen der Baukästen gibt es von LEGO® keine Informationen. Bisher wurde von dieser Seite nur veröffentlicht, dass in den ersten drei Verkaufsmonaten Ende 1998 in den USA ungefähr 80.000 Exemplare verkauft wurden. Näheren Aufschluss über die wirkliche Verbreitung dieser Robotersysteme geben die Aussagen von Mindstorms™ Experten: so vermutet *Jonathan Knudsen*[5], dass in den Jahren 1999 und 2000 jeweils »einige hunderttausend« Einheiten verkauft wurden[6]. Anfang 2001 spricht *Joe Nagata*[7] von 500.000 verkauften Baukästen in nur einem Jahr[8]. Von daher ist davon auszugehen, dass heute, gut fünf Jahre nach dem ersten Erscheinen, weltweit über eine Million dieser Systeme existieren.

Trotz dieser großen Stückzahl und einer riesigen Fangemeinde ist bisher keine Softwareanwendung erschienen, die es ermöglicht, das Verhalten eines typischen Mindstorms™ Modells an einem PC zu simulieren. Es existieren zwar Programme mit Namen wie legoSim oder emuLegos, jedoch beschränken sich diese auf eine Simulation der grundlegenden Ein- Ausgabefunktionen, wobei in keinem Fall mit den gewohnten Programmierumgebungen gearbeitet werden kann. Die graphische Simulation von autonomen, mobilen Robotermodellen ist mit keinem dieser Produkte auch nur ansatzweise möglich.

---

[5] *Jonathan Knudsen* lebt in Princeton, New Jersey, USA. Er ist der Verfasser des Buches »Das inofizielle Handbuch für LEGO® Mindstorms™ Roboter«.

[6] Vgl. Knudsen, NZZ Folio 12/2000, o.S.

[7] *Joe Nagata* lebt in Japan. Er ist Computergraphik- und Digitalkünstler und Verfasser des Buches »Joe Nagata's LEGO® Mindstorms Idea Book«.

[8] Vgl. Nagata, Mindstorms Idea Book, o.S.

1.3 Das Projekt Roberta am Fraunhofer Institut AIS

Das Projekt Roberta hat zum Ziel, bei Mädchen und Frauen Interesse für Informatik, Ingenieur- und Naturwissenschaften zu wecken und Verständnis für technische Systeme zu fördern. Dafür werden Roboterkurse entwickelt, die auch Mädchen ansprechen. Diese werden als Teil des Bildungsangebots etabliert.

Die Informationen in diesem Abschnitt basieren im Wesentlichen auf zwei Dokumenten, die das Projekt näher beschreiben: zum einen das Roberta Projektblatt[9] und zum anderen die Informationsbroschüre für Lehrkräfte[10]. Es wird im Folgenden darauf verzichtet, auf diese Dokumente wiederholt zu verweisen. Stattdessen wird an dieser Stelle darauf hingewiesen, dass große Teile dieses Abschnitts sinngemäß und teilweise auch wörtlich aus diesen Dokumenten übernommen wurden.

1.3.1 Motivation

Schon heute bestimmen technische Systeme unser tägliches Umfeld. Da deren Einsatz in Zukunft weiter zunehmen wird, sind Informatik, Mechatronik und Robotik Schlüsselbereiche für die gesamte Volkswirtschaft. Entgegen ihrer Bedeutung ist das Interesse an technischen Themen in unserer Gesellschaft eher gering, was zu einem akuten Nachwuchsmangel in technischen Berufen führt. Das Interesse ist bei Mädchen noch geringer als bei Jungen. In Deutschland betrug der Anteil von Frauen in technischen Studienfächern im Wintersemester 2001/2002 im Bereich Ingenieurwissenschaften ca. 21%, im Bereich Elektrotechnik sogar nur 9%[11].

---

[9] Müllerburg et al., Roberta Projektblatt

[10] Müllerburg et al., Roberta – Informationen zum Projekt

[11] Quelle: Statistisches Bundesamt

Roboter und Roboterbaukästen bieten eine hervorragende Grundlage, um junge Menschen an technische Themen heran zu führen. Mit ihnen lassen sich nicht nur Themen der Robotik vermitteln. Sie bieten darüber hinaus einen attraktiven Zugang zu Informatik und Technik im Allgemeinen. Dies gilt für alle Altersgruppen und jeden Stand des Vorwissens. Erfahrungen im Vorfeld des Projekts zeigten deutlich, dass Roboter auch für Mädchen und Frauen attraktiv sind.

1.3.2 Ziele und Ergebnisse

Im Projekt Roberta werden Lehr- und Lernmaterialien erarbeitet, die KursleiterInnen die Möglichkeit geben, Roboterkurse mit vertretbarem Aufwand selbständig durchführen zu können. Parallel dazu wird eine Organisationsstruktur aufgebaut, die sicherstellt, dass KursleiterInnen bei Bedarf in ihrer Region die notwendige Unterstützung finden und dass die Ergebnisse auch nach Abschluss des Projekts verwendet und weiterentwickelt werden können. Zu diesem Zweck wird ein Netzwerk regionaler Zentren mit einer zentralen Koordinierungsstelle aufgebaut, deren Sitz am Fraunhofer Institut AIS in Sankt Augustin ist.

Zur Sicherstellung der Qualität der Ergebnisse ist die Universität Bremen mit der Begleitforschung beauftragt. Die Entwicklung und Evaluierung der Roboterkurse soll zu Erkenntnissen darüber führen, wie sie gestaltet sein müssen, damit sie nicht nur für Jungen, sondern auch für Mädchen interessant sind. Für die qualitative Erhebung werden Interviews mit KursleiterInnen durchgeführt und Kurse beobachtet. Zum Zwecke der quantitativen Erhebung werden verschiedene Fragebögen bereitgestellt, die von KursleiterInnen und TeilnehmerInnen ausgefüllt werden.

Die ersten Auswertungen der quantitativen Erhebung führen zu deutlich positiven Ergebnissen. Nach Auswertung von 250 Fragebögen, die von KursteilnehmerInnen ausgefüllt wurden, lässt sich unter anderem folgendes feststellen:

- 93% hat die Teilnahme Spaß gemacht
- 83% würden den Kurs weiterempfehlen
- 63% würden gern an einem weiterführenden Kurs teilnehmen.

Bereits nach einem zweistündigen Schnupperkurs können sich die Mädchen eher als vor dem Kurs vorstellen, Computer-Expertin zu werden, wenn sie es wirklich wollen.

1.3.3 Roboter und Roboterbaukästen

Roboter bieten einen im Gegensatz zum Internet technikbezogenen Zugang zur Informationstechnologie. Die Vielfältigkeit der Robotik ermöglicht die Einführung in viele Teilgebiete. Softwaretechnik und Rechnerarchitektur können ebenso angesprochen werden wie wichtige Grundlagen der Mathematik und Physik. Dabei werden fachliche Kenntnisse und Methodenwissen erworben und geübt. Die erworbenen Kenntnisse, Fähigkeiten und Erfahrungen können auch in anderen Lern- und Lebensbereichen genutzt werden.

Auch Mädchen und Frauen können motiviert werden, sich mit technischen Themen zu beschäftigen. Allerdings müssen Sinn und Nutzen erkennbar sein, z.B. durch Anwendungsbezug (nützliche, vielleicht sogar Leben rettende Maschinen) oder berufliche Perspektiven.

Die Attraktivität der Roboter hilft, Hemmschwellen zu überwinden und der spielerische Umgang mit ihnen fördert den Abbau von Skepsis gegenüber Technik. Die Faszination daran, ein technisches System selbst von Grund auf zu entwickeln, weckt Interesse und Lernbereitschaft. Indem Mädchen und Frauen Technik selbst gestalten, wird ihr Selbstbewusstsein gestärkt.

1.3.4 Roboterkurse

Es gibt bisher kein Schulungsmaterial, das Mädchen oder Frauen gezielt anspricht. Typische Roboterbeispiele wie Panzerfahrzeuge und Kampfroboter haben hier eine eher abschreckende Wirkung. Es besteht die Gefahr einer Entwicklung wie sie offensichtlich bei den Computerspielen zu beobachten ist: sie sind von den Interessenschwerpunkten der Jungen geprägt und stoßen bei Mädchen vielfach auf Desinteresse oder gar Ablehnung[12].

Hier setzt das Projekt Roberta an und entwickelt Roboterkurse, welche die Zielgruppe der Mädchen besonders berücksichtigen, wobei ein Mindestalter von 10 Jahren vorausgesetzt wird. Die naturwissenschaftlichen und technischen Interessen beider Geschlechter werden hierbei gefördert.

Im Mittelpunkt der Einführungskurse stehen Experimente wie »Hindernisse vermeiden« und »Spur fahren«. Fachliches Wissen, z.B. über Zahnräder, Antriebe, Sensoren oder Programmierung, kann anhand der Experimente vermittelt werden. In längeren Kursen bauen die Experimente aufeinander auf, sodass die TeilnehmerInnen schrittweise von kleinen, einfachen Aufgaben zu komplexen Aufgaben geführt werden. Dabei ist ein wichtiges Ziel des Projektes, Themen für Experimente zu finden, die insbesondere für Mädchen attraktiv sind.

1.4 Nutzen der Mindstorms™ Simulation

In den im Rahmen des Projekts Roberta durchgeführten Schulungen wurde seitens der teilnehmenden Lehrkräfte wiederholt der Wunsch nach einer Simulation artikuliert. Obwohl außer Frage steht, dass die Erfahrung am »echten« Roboter durch nichts zu ersetzen ist, gibt es

---

[12] Vgl. Gorriz, C.M.; Medina, C: Engaging Girls with Computers through Software Games, S. 42-49

mehrere Punkte, die für den partiellen Einsatz einer Simulation sprechen:

- Die Baukästen sind nicht stetig verfügbar. Wenn das Bedürfnis besteht, Programmiererfahrung zu sammeln, um auf einen Kurs vorbereitet zu sein, bietet eine Simulation gute Übungsmöglichkeiten.
- Bei Kursen mit vielen (über zwanzig) TeilnehmerInnen ergibt sich aufgrund der optimalen Arbeitsgruppengröße von jeweils zwei bis maximal drei Personen zwangsläufig ein Problem: nicht alle Einrichtungen verfügen über zehn oder mehr Roboterbaukästen. Hier kann es bei entsprechender PC-Ausstattung eine gute Lösung sein, die Einarbeitung und Vorbereitung der Experimente am Simulator durchführen zu lassen und anschließend die Ergebnisse an dem oder den vorhandenen realen Modellen zu testen.
- Eine Simulation kann auch genutzt werden, um die Fehlersuche in Programmen zu vereinfachen. Häufig werden unerwartete Phänomene mit Technikproblemen erklärt. Verhält sich ein Roboter nicht so, wie vom Programmablauf her eigentlich erwartet, werden schnell Probleme wie niedrige Batteriespannung, kaputte Motoren oder Sensoren, bis hin zum Ausfall des gesamten Mikroprozessorsystems in Betracht gezogen. Der Test am Simulator kann physikalische Probleme und Einflüsse ausschließen und somit helfen, das Problem zu analysieren.
- Mittels einfacher Dateiauswahl sind komplexe Simulationsumgebungen schnell verfügbar. Dies stellt gerade im fortgeschrittenen Experimentbereich einen Vorteil dar. So ist z.B. die Bereitstellung eines realen Riesenlabyrinths in der Praxis ein echtes Problem.

2 Roboter und deren Simulation

2.1 Technische Grundlagen

In der VDI-Richtlinie 2860 wurde der Begriff Roboter schon 1982 folgendermaßen definiert: »Ein Roboter ist ein universell einsetzbarer Bewegungsautomat mit mehreren Achsen, dessen Bewegungen hinsichtlich Folge und Wegen bzw. Winkeln frei programmierbar und gegebenenfalls sensorgeführt sind«.[13]

Diese Definition deckt heute sicher nicht mehr alle Bereiche der Robotik ab, besitzt aber für die meisten Robotersysteme immer noch Gültigkeit.

Robotergenerationen

In der Entwicklung der Roboter unterscheidet Dieter W. Wloka 1992 die folgenden drei so genannten Robotergenerationen[14]:

- 1. Generation (1975) – Kennzeichen: Anfahren fester Haltepunkte, keine Sensoren, einfache Aufgaben, exakte und robotergerechte Umgebung erforderlich
- 2. Generation (1980) – Kennzeichen: verschiedene Bewegungsverfahren sind vorhanden, Einsatz eines Steuerrechners, jeweils ein eigener Prozessor für jede Roboterachse, der Roboter kann mit Hilfe einer (höheren) Programmiersprache programmiert werden, Sensorfunktionen sind in die Sprache eingebunden
- 3. Generation (Zukunft) – Kennzeichen: Multisensorik vorhanden, adaptives Verhalten durch Methoden der »Künstlichen Intelligenz«, aufgabenorientierte und implizite Programmierung – auch autonome und mobile Roboter

---

[13] VDI-Richtlinie 2860 – Handhabungsfunktionen

[14] Vgl. Wloka, Dieter W., Robotersysteme, Band 1, S. 4-5

Was Wloka vor mehr als zwanzig Jahren noch unter »Zukunft« abhandelte, ist heute längst Realität geworden. Wissenschaftler in der ganzen Welt arbeiten daran, autonome und mobile Roboter zu entwickeln, wobei die Kennzeichen der oben genannten 3. Generation mittlerweile selbstverständlich geworden sind.

Mindstorms™ Roboter sind schwer in dieses Schema einzuordnen. Sie haben sowohl Kennzeichen der 2. als auch der 3. Generation von Robotern. Sie können mit höheren Programmiersprachen programmiert werden, die auch Sensorfunktionen enthalten. Sie werden von einem Rechnersystem gesteuert und sind, je nach Modell, autonom und mobil. Im Gegensatz zu ihren »großen Brüdern« haben sie jedoch keinen eigenen Prozessor (auch Controller genannt) für jede Achse. Weiterhin ist die Programmierung von adaptivem, intelligentem Verhalten aus Gründen der geringen Speicherplatzverfügbarkeit kaum möglich.

Das besondere an den Mindstorms™ Robotern ist deren gute Handhabbarkeit. Der steuernde, hochintegrierte Baustein hat ein geringes Eigengewicht und ist sehr kompakt. Weitere Komponenten wie Sensoren oder Motoren lassen sich problemlos per Steckverbindung anschließen.

Aufbau eines Robotersystems

Ein Robotersystem wird normalerweise in die folgenden drei Hauptbereiche gegliedert:

- Mechanik
- Sensorik
- Steuerung

Diese drei Hauptbereiche lassen sich, je nach Anwendungsgebiet, feiner unterteilen, was jedoch den Rahmen dieser Arbeit sprengen würde.

Roboterachsen

Eine Roboterachse ist ein angetriebenes und geführtes mechanisches Glied. In den meisten Fällen kommen ausschließlich rotatorische und translatorische Roboterachsen zum Einsatz, wobei jede Achse im Allgemeinen einen Freiheitsgrad repräsentiert.

Kinematik

Der Begriff der Kinematik wird von Wloka folgendermaßen beschrieben: »Unter der Kinematik eines Roboters versteht man die Roboterachsen und alle weiteren mechanischen Teile, mit Ausnahme der Antriebe. Der Begriff ist deshalb hier nicht im Sinne der Physik zu verstehen«.[15]

Effektoren

Der Effektor ist die Funktionseinheit, welche die eigentliche Handhabungsaufgabe durchführt, wie z.B. das Greifen und Bearbeiten von Handhabungsobjekten. Prinzipiell werden zwei Typen von Effektoren unterschieden: Werkzeuge und Greifer.

Bei den Werkzeugen ist die Anzahl der möglichen Formen nahezu unbegrenzt. Häufig verwendete Werkzeuge im Industrieroboterbereich sind z.B. Werkzeuge für die mechanische Bearbeitung, für das Schweißen oder für die Oberflächenbeschichtung.

Die Greifer lassen sich wiederum feiner unterteilen. Von einfachen Greifern mit Greiferbacken über Greifer an Wechselsystemen oder Mehrfachgreifer bis hin zu so genannten »geschickten Händen«.[16]

Auch mit Mindstorms™ Bauteilen ist die Konstruktion von Greifern möglich. Die folgende Abbildung zeigt einen solchen Greifer.

---

[15] Wloka, Dieter W., Robotersysteme – Band 1, S. 6

[16] Vgl. Wloka, Dieter W., Robotersysteme – Band 1, S. 7

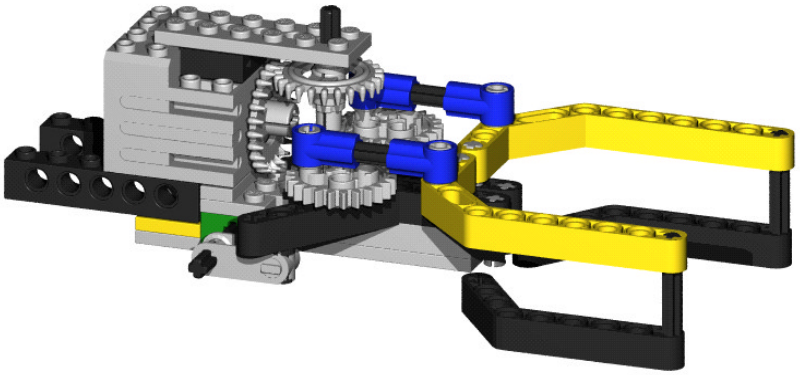

Abbildung 1: Ein Mindstorms™ Greifer

Antriebe

Roboterachsen und Effektoren können sich aus eigener Kraft nicht bewegen. Aus diesem Grund werden Antriebe eingesetzt, die die oben genannten Teile bewegen. Es werden folgende Antriebsarten unterschieden: pneumatische Antriebe, hydraulische Antriebe und elektrische Antriebe. Oft werden so genannte Mischformen eingesetzt. Jeder Antrieb kann in folgende Funktionsgruppen unterteilt werden: Energieversorgung, Servoeinheiten, Antriebsmotor, Bremsen und Kraftübertragungselemente. Jeder Antrieb muss mit Energie versorgt werden. Die Servoeinheiten haben die Aufgabe eines Wandlers: sie wandeln die Signale der Steuerung in Signale für die Arbeitseinheiten um. Durch die Antriebsmotoren werden Antriebskräfte und –momente erzeugt. Die Bremsen haben die Aufgabe die Roboterachsen nach einer Bewegung zu verlangsamen und werden auch für den Stillstand des Robotersystems benötigt. Kraftübertragungselemente werden dann eingesetzt wenn beispielsweise Motoren nicht direkt oder unmittelbar an Roboterachsen angebracht werden können.

Sensoren

Sensoren decken verschiedene Aufgabenbereiche ab: sie erfassen beispielsweise die Zustände des Handhabungsobjektes, sie messen physikalische Größen, sie identifizieren und bestimmen die Lage von

Werkstücken (Mustererkennung) und vieles mehr. Die Entwicklung immer neuer, intelligenter Sensoren spielt in der Robotik eine wichtige Rolle.

Steuerung

Die Steuerung besteht klassischerweise aus einem Rechnersystem mit Bedienungskonsole sowie einem Programmierhandgerät. Die Funktion der Steuerung ist folgende: Steuerung und Überwachung des Arbeitsablaufs und Austausch von Signalen mit Peripheriegeräten. Das Rechnersystem dient zur Durchführung des Steuerprogramms, oft auch zur Unterstützung der Steuerprogrammentwicklung, zur Verarbeitung der Sensordaten und zur Kommunikation mit anderen Systemen.

2.2 Simulation von Robotersystemen

Der Begriff »Simulation« ist vieldeutig und führt zu verschiedenen Assoziationen. Schon vor mehr als zwanzig Jahren stellte u.a. Susan L. Solomon die enorme Fragmentierung dieser Disziplin sowohl im Anwendungs- als auch im Forschungsbereich fest[17]. Durch neuere Entwicklungen im Bereich der Informatik ist diese Zersplitterung weiter vorangetrieben worden. Schon 1994 erklärt F. Liebl: »Simulation wird heute im Dunstkreis der Schlagworte *Cyberspace*[18] und *3D-Animation*[19] ebenso gebraucht wie in Zusammenhang mit neuen

---

[17] Solomon, Susan L., Building Modelers – Teaching the Art of Simulation, S. 65-72.

[18] Der Begriff *Cyberspace* bezeichnet künstlich erschaffene Welten, die meist interaktiv (z.B. mit 3D-Brille und Datenhandschuh) erfahrbar sind.

[19] *3D-Animation* ist die dreidimensionale Darstellung (Projektion) von Objekten und deren Bewegung mit Hilfe von Berechnungen.

Kommunikationsformen, z.B. *Computer-Based Training*[20] oder [...] Multimedia«.[21]

Eine genaue Definition des Begriffs Simulation findet sich in der VDI-Richtlinie 3633: »Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind«.

Ein Simulator muss andere Systeme nicht gänzlich nachbilden, sondern nur deren (zu erwartendes) Verhalten in einem festgelegten Modellrahmen darstellen.

Schon vor der Entwicklung von modernen, publikumswirksamen Simulationstechnologien wurden Simulationen eingesetzt, so etwa Flugsimulatoren zum Trainieren von Piloten. Derartige Formen der Simulation werden als Echtzeitsimulation bezeichnet, da sie zum Ziel haben, reale Vorgänge zeitlich identisch abzubilden. Auch die Simulation von Robotersystemen wird typischerweise in Form einer Echtzeitsimulation realisiert, wobei meist wissensbasierte Verfahren mit graphischen Verfahren gekoppelt werden.

Neben diesem Bereich existiert das weite Feld der Simulation diskreter Ereignisse: »Ziel ist in diesem Fall nicht, das Geschehen zeitlich identisch abzubilden und den Benutzer unmittelbar an den Vorgängen teilhaben zu lassen. Es soll vielmehr – in Analogie zu den früher üblichen militärischen Sandkastenspielen – versucht werden, im vorhinein eine Vorstellung über Geschehnisse und Abläufe in einem bestimmten Weltausschnitt, dem Realsystem, zu erlangen«.[22] Eine nähere Betrachtung der Simulation diskreter Ereignisse erfolgt im

---

[20] *Computer-Based Training* ist ein allgemeiner Begriff für moderne Lernformen, bei denen nicht ein Mensch, sondern ein Computer den Lernablauf moderiert.

[21] Liebl, Franz, Simulation, S. 3.

[22] Liebl, Franz, Simulation, S. 3

Rahmen dieser Arbeit nicht, da sie im Zusammenhang mit Robotersystemen keine Rolle spielt.

Wissensbasierte Simulation

Es gibt zwei Methoden zur Vorhersage des Verhaltens eines Systems: das numerische Simulationsverfahren und das wissensbasierte Verfahren. Mit numerischen Simulationen meint man mathematische Gleichungen oder Wahrscheinlichkeitsverteilungen. Wissensbasierte Verfahren basieren auf Regeln. Numerische Verfahren können im Gegensatz zu den wissensbasierten Verfahren die Ergebnisse nicht erklären.

Verschiedene Verfahren und Systeme werden zurzeit bei mobilen Robotern kombiniert. Schwerpunkte hierbei sind die Entwicklung von Steuerstrukturen, Programmierverfahren, Weltmodellen, etc. Die Fähigkeit zu Lernen ist für zukünftige Systeme von sehr großer Bedeutung.

Graphische Simulation

Als Computeranimation wird im Allgemeinen die Gestaltung von Bewegungsabläufen mit Hilfe eines Computers verstanden. Die eingesetzte Technik kann neben Robotersimulationen auch für andere Bereiche verwendet werden: Erzeugung von Zeichentrickfilmen, Generierung fotorealistisch aussehender künstlicher Objekte, Erzeugung künstlicher Welten und vieles mehr. Die Animation bietet demnach die Möglichkeit technische Inhalte visuell darzustellen.

3D-Animationen lassen sich durch Drahtrahmen oder Festkörpermodellen darstellen. Hierbei ist die simultane Darstellung aller Bewegungsabläufe erforderlich. Des Weiteren sollte ein Simulationssystem auch die Möglichkeit einer Kollisionsprüfung beinhalten. Sensorfunktionen müssen bei Simulationen ebenso »eingebaut« werden.

2.3 Typische Fehler bei der Simulationsentwicklung

Es ist bekannt, dass nicht jede in Auftrag gegebene Studie zum Erfolg führt und die Erwartungen des Auftraggebers erfüllt. Unmittelbare Anlässe für das Scheitern sind im Normalfall Kosten- und Zeitüberschreitungen sowie die Unzufriedenheit des Auftraggebers mit der Aussagefähigkeit der Ergebnisse, wobei die tieferen Gründe für Misserfolge sich häufig ähneln[23]. Im Folgenden sollen daher häufig auftretende, jedoch vermeidbare Fehler zusammengefasst dargestellt werden.

Falsche Definition des Studienziels

Ausgangs- und Endpunkt eines Simulationsprojekts muss immer das Realproblem sein. Es muss zwischen den Auftraggebern und dem Entwickler(-team) vereinbart werden, welche Erkenntnisziele mit dem Simulationsprojekt verfolgt werden sollen. Wichtig ist, dass Modellierung und Simulation nicht als Selbstzweck betrieben werden. »Was wollen wir durch das Simulationsmodell erfahren? Wie lautet die konkrete Entscheidungssituation? [...] Auf diese und ähnliche Fragen haben sich die Parteien zu einigen«.[24]

Ungenügende Partizipation der Auftraggeber

Poole/Szymankiewicz erkannten schon 1977 die folgenden Zusammenhänge: Die Dokumentation des Studienziels allein reicht für ein erfolgreiches Projekt nicht aus. Auftraggeber und Entwickler müssen während der gesamten Dauer der Arbeit in enger Kommunikation und Kooperation miteinander stehen. Nur regelmäßige Reviews, in denen Zwischenergebnisse präsentiert und diskutiert werden sowie kurzfristige Ziele gesetzt werden, liefern die notwendige Rückkopp-

---

[23] Vgl. Liebl, Franz, Simulation, S. 222

[24] Vgl. Liebl, Franz, Simulation, S. 223

lung und sorgen dafür, dass falsch eingeschlagene Wege frühzeitig korrigiert werden können[25].

»Entsprechend ungünstig ist daher das Umfeld, wenn eine enge Kooperation mit dem Kunden nicht möglich ist – beispielsweise, weil der Kunde glaubt, vom Tagesgeschäft überlastet zu sein oder von dem Problem nicht viel zu verstehen, und sich deshalb frühzeitig aus dem Projekt zurückzieht«.[26]

Ungeeigneter Detaillierungsgrad

»Small models, small troubles« - Reitman prägte dieses Sprichwort, das in mehrfacher Hinsicht ernst genommen werden sollte[27]. Es ist sogar so, dass große Modelle im Allgemeinen überproportional viele Probleme bereiten. Neben dem enormen Entwicklungsaufwand stehen in diesem Fall auch hohe Validierungszeiten. Liebl stellt hierzu fest: »Ist das Modell endlich hinreichend glaubwürdig, wird mit Sicherheit auch die Durchführung der Ergebnisläufe zur harten Geduldsprobe«.[28]

Wahl der falschen Entwicklungswerkzeuge

Auf die hinlänglich bekannte Diskussion um die am Markt konkurrierenden Werkzeuge soll hier nicht näher eingegangen werden. Statt dessen soll grundsätzlich erläutert werden, welche Punkte bei der Entwicklung einer Echtzeit-Simulation beachtet werden sollten.

- Die benutzte Programmiersprache muss die Fähigkeit besitzen, mehrere parallel ablaufende *Threads*[29] zu erzeugen.

---

[25] Poole, T.G.; Szymankiewicz, J.Z., Using Simulation to Solve Problems, S. 237

[26] Liebl, Franz, Simulation, S. 224

[27] Reitman, J., Computer Simulation Applications, S. 356

[28] Liebl, Franz, Simulation, S. 226

[29] *Threads* sind eine Sonderform von Prozessen. Bei umfangreichen Betriebssystemen wie z.B. Windows™ und Linux wird zwischen Prozessen, die einen eigenen

- Das erzeugte Programm muss bei der Ausführung möglichst effizient und vor allem mit konstanter Geschwindigkeit laufen.

- Bezüglich des Programmierstils wird die Benutzung von selbsterklärenden Klassen-, Methoden- und Variablenbezeichnern dringend empfohlen. So bemerkt Liebl: »Eine Ersparnis, die sowohl kurzfristig als auch bei Langzeitprojekten auftritt, resultiert aus der (technischen) Dokumentation der Programme. Es wäre übertrieben, zu behaupten, [...] -sprachen mit einer nahe an der Schriftsprache liegenden Syntax wären absolut selbstdokumentierend. Jedoch sieht man sofort, was das Programm an einer bestimmten Stelle ausführt und braucht im Zweifelsfall nur zu ergänzen, wieso dies geschieht. Derart ausgestattet kann man nach unserer Erfahrung auch noch Jahre später die Zusammenhänge in endlicher Zeit rekonstruieren und gegebenenfalls Anschlussprojekte durchführen«.[30]

- Die Benutzung von Werkzeugen für die Realisierung von physikalischem Verhalten oder die graphische Ausgabe der Simulationsumgebung ist dringend zu empfehlen. Es existiert eine Vielzahl von Tools, die teilweise auch frei verfügbar sind. In diesem Bereich »das Rad neu zu erfinden« ist in den meisten Fällen vertane Zeit. Eine Ausnahme bilden in diesem Zusammenhang High-End Simulationen wie z.B. diverse Computerspiele oder auch Trainingsgeräte für Piloten.

Speicherbereich zugewiesen bekommen und solchen, die sich einen gemeinsamen Bereich teilen, unterschieden.

[30] Liebl, Franz, Simulation, S. 227

3 Pflichtenheft

Das Pflichtenheft beschreibt die allgemeinen Anforderungen und legt den genauen Rahmen des zu entwickelnden Produktes fest. Einsatzbedingungen sowie Hard- und Softwarevoraussetzungen werden in Kapitel 7 behandelt.

3.1 Anforderungen an die Mindstorms™ Simulation

Oberste Priorität bei der Entwicklung der Mindstorms™ Simulation hat die Benutzungsfreundlichkeit. Das Produkt soll von LehrerInnen, SchülerInnen und ganz allgemein Menschen ohne besondere Informatikkentnisse benutzt werden können. Die beste Simulation macht in diesem Fall keinen Sinn, wenn sie nicht intuitiv benutzbar ist.

Dem entsprechend soll das Robotermodell mit den auch real üblichen Programmierumgebungen gesteuert werden können. Den AnwenderInnen soll kein zusätzlicher Lernaufwand bezüglich der Programmierung der Roboter entstehen.

Weiterhin müssen sowohl Simulationsumgebungen als auch Robotermodelle bereitgestellt werden, die denen der realen Roberta-Kurse entsprechen.

3.2 Zielbestimmung und Kriterien

Die Simulation soll die Möglichkeit bieten, Experimente mit LEGO® Mindstorms™ Robotern durchzuführen, auch wenn die normalerweise dazu benötigte Hardware (RCX, Infrarot-Transmitter, Sensoren, Motoren und weitere Bauteile) nicht verfügbar ist. Zu diesem Zweck wird eine Simulationsumgebung bereitgestellt, die das Verhalten und die Funktionalität der nicht vorhandenen Hardware sowohl technisch emuliert als auch graphisch präsentiert. Besonderer Wert wird hierbei auf die Benutzungsfreundlichkeit der Anwendung gelegt.

3.2.1 Musskriterien

Folgende Funktionalitäten sind zentral und unverzichtbar:

- Es wird eine graphische Umgebung bereitgestellt, in der das Verhalten eines Robotermodells verfolgt werden kann. Dieses kann aus verschiedenen vorgegebenen Robotermodellen ausgewählt werden. Grundsätzlich wird ein sich fortbewegender (autonomer) Roboter mit verschiedenen Sensorausstattungen unterstützt (in Anlehnung an den LEGO® Roverbot).

- Unterstützt werden die Ansteuerung der Motoren, die Abfrage von Berührungs- und Lichtsensoren, der Umgang mit Timerfunktionen und Variablen, die Ausgabe von Tönen und Systemklängen, sowie die gesamte Programmablaufsteuerung einschließlich mehrerer paralleler Tasks.

- Die Simulation bildet den Umgang mit dem RCX softwareseitig ab, sodass die vier Bedienungsknöpfe wie üblich benutzt werden können.

- Die Flüssigkristall-Anzeige wird so dargestellt, wie es auch real zu erwarten wäre.

- Die im Simulator auszuführenden Programme werden in Form des so genannten RCX-Bytecode aus den Programmierumgebungen übernommen, die diese Art von Code erzeugen. Zurzeit sind solche Umgebungen das »Bricx Command Center« (Programmiersprache: NQC) und das »LEGO® Robotics Invention System™« (graphische Programmiersprache: RCX-Code).

- Beliebige Programme im RCX-Bytecode können auf dem Simulator ausgeführt werden. Der Bytecode Interpreter, der normalerweise in der LEGO® Standard-Firmware enthalten ist, wird softwareseitig emuliert. Bytecodebefehle, die von der Simulation nicht unterstützt werden, führen nicht zu einem Programmabsturz, sondern werden sinnvoll abgefangen.

- Die Übertragung der mit den oben genannten Umgebungen erstellten Programme findet auf dem Dateiweg statt. Die Programme werden wie üblich abgespeichert und können dann vom Simulator aus geladen werden.
- Die Ansicht der Simulationsumgebung wird dreidimensional dargestellt, wobei der Sichtwinkel und der Zoomfaktor per Maussteuerung änderbar sind. Auch das Robotermodell ist positionierbar und drehbar.
- Es werden mehrere vorgegebene Simulationsumgebungen bereitgestellt, zwischen denen (per Menüsteuerung) umgeschaltet werden kann.
- Es werden mehrere vorgegebene Robotermodelle bereitgestellt, zwischen denen (per Menüsteuerung) umgeschaltet werden kann. Verfügbar sind Modelle mit einem und zwei Berührungssensoren sowie Modelle mit einem oder zwei Lichtsensoren.

3.2.2 Wunschkriterien

Mögliche Erweiterungen, die nicht in der ersten Version erstellt werden:

- Unterstützung von Rotationssensoren
- Unterstützung von weiteren selbst entwickelten Sensoren
- Unterstützung von mehreren Robotern in einer Simulationsumgebung mit der Möglichkeit zum Austausch von Infrarot-Nachrichten
- Simulationsumgebungs-Editor
- Robotermodell-Editor
- Zeitraffer-, Zeitlupen- und Vor- Zurückmodi für den Simulationsablauf

- Übertragung der Programme mit den üblichen Schaltflächen direkt aus den Programmierumgebungen
- Unterstützung der Benutzung des »Datalog«

3.2.3 Abgrenzungskriterien

Funktionalitäten, die in dem Projektrahmen nicht vorgesehen sind und von daher von dem Produkt nicht geleistet werden:

- Das Produkt stellt keine komplette Emulation der RCX Hardware zur Verfügung, sondern bildet nur das Verhalten eines RCX bei der Ausführung von RCX-Bytecode nach. Somit können auch keine alternativen Betriebssysteme für den RCX (leJOS, brickOS, etc.) mit dem Produkt benutzt werden.
- Das Produkt stellt keine Programmierumgebung bereit. Programme für die Roboter werden mit externer, RCX-Code erzeugender Software (z.B. RIS, NQC) erstellt.

3.3 Produktschnittstellen

Produktschnittstellen nach außen (Export) sind zurzeit nicht vorgesehen.

Für den Import der RCX Bytecode-Programme gibt es theoretisch mehrere Möglichkeiten, wobei nur die folgenden beiden hier genannten in der ersten Version realisiert werden:

- Öffnen einer speziellen, vom NQC Compiler erzeugten Bytecodedatei (alle Betriebssysteme)
- Öffnen einer speziellen, von der RIS-Software erzeugten Skriptdatei (nur Windows™)

In weiteren Versionen möglich:

- Übertragen der Programme mit den externen Programmierumgebungen »Robotics Invention System™« oder »Bricx

Command Center«. Diese Programme werden über eine emulierte serielle Schnittstelle in die Simulation importiert (nur unter Windows™).

3.4 Produktfunktionen

Die BenutzerInnen sollen die folgenden Funktionen auswählen können.

In der ersten Version:

- Simulationsumgebung auswählen
- Robotermodell und dessen Ausstattung auswählen
- NQC Bytecode aus Datei importieren
- RIS Bytecode aus Datei importieren
- RCX spezifische Funktionen (View, Prgm, On-Off und Run)
- Informationen über die Simulation (Versionsnummer, Autor, etc.)

In weiteren Versionen:

- Neue Simulationsumgebung erstellen u. speichern
- Neues Robotermodell entwerfen u. speichern
- Mehrere Modelle gleichzeitig starten und anhalten
- Funktionen zur Betrachtung und Speicherung des »Datalog«

3.5 Produktleistungen

Hier werden die Leistungen der Simulation im laufenden Betrieb (Echtzeitverhalten) festgehalten:

Das Produkt stellt das Verhalten eines Robotermodells graphisch in Echtzeit dar. Die Bewegung des Modells in seiner Umgebung kann visuell verfolgt werden, wobei ein Höchstmaß an Realität bezüglich

Beschleunigung, Geschwindigkeit und Ausrichtung des Modells in seiner Umgebung eingehalten wird.

Angestrebt wird eine »ruckelfreie« Bewegungsdarstellung. Ein Bildaufbau mit fünfzig *Frames*[31] pro Sekunde reicht im Allgemeinen dafür aus.

Die Größe des Darstellungsfensters ist frei skalierbar, wobei auch der Fensterinhalt skaliert wird.

3.6 Betriebssysteme

Das Produkt soll plattformunabhängig implementiert werden und für die Abnahme (in der ersten Version) unter Windows™ Systemen lauffähig sein.

3.7 Testfälle

Als Testfälle dienen die im Projekt Roberta eingesetzten »einfachen Aufgaben«, z.B. der »Hindernisvermeider«, »eingesperrt im Kreis«, und der »Linienfolger«.

---

[31] Der Begriff *Frame* bezeichnet im Zusammenhang mit Computeranimation ein komplett dargestelltes Einzelbild in einer animierten Bildfolge. Die Anzahl der *Frames* pro Sekunde beeinflusst, wie »flüssig« eine Bewegung wahrgenommen wird.

4 Entwurfsentscheidungen und Werkzeuge

4.1 Komponenten

Sinnvollerweise werden vor Beginn der Entwicklung die zu realisierenden Komponenten analysiert.

Grundsätzlich ergeben sich folgende Hauptaufgaben der Simulation:

- Import der Roboterprogramme
- *Emulation*[32] des RCX
- Berechnung und Darstellung des Roboterverhaltens

Homogene Softwareprodukte sind zwar leichter zu implementieren als modulare Systeme, haben jedoch im Zusammenhang mit Echtzeitsimulationen einen entscheidenden Nachteil: die gesamte Rechenlast liegt bei einem System. Es wäre bei einem homogenen Entwicklungsansatz nicht möglich, die benötigte Rechenzeit für die Emulation des RCX von einem anderen Computer bereitstellen zu lassen als dem, der die Rechenleistung für die Berechnung der Umgebungsphysik und die graphische Ausgabe bereitstellt. Von daher wird das Produkt grundsätzlich modular aufgebaut, wobei das in Abbildung 2 dargestellte Schema eingehalten wird. Der RCX Emulator ist für den Import der Roboterprogramme zuständig. Wenn ein Roboterprogramm läuft, übergibt es die jeweils aktuellen Motorzustände an das für die physikalischen Berechnungen zuständige

---

[32] Laut Definition bedeutet *Emulation*, dass ein Computer oder Prozessor sich wie ein anderer verhält. Von daher ist *Emulation* der Versuch, einen Computer mit einem anderen Computer oder Programm nachzubilden. Ein Emulator ist ein Programm, eine Software oder Hardware, die es einem Computer erlaubt, Befehle oder Programme auszuführen, die für einen anderen Typ von Computer oder Prozessor entwickelt wurden. Ein Computer Emulator ist ein Programm, welches das Verhalten anderer Computersysteme nachahmen kann.

Modul, welches wiederum die ermittelten Sensorwerte zurückliefert und die graphische Ausgabe leistet. Eine weitere Modularisierung ist an dieser Stelle durchaus denkbar, wird aber nicht vorgenommen, da die Berechnung des Roboterverhaltens und die graphische Ausgabe beide von der ODE übernommen werden und dort eng verzahnt sind (siehe dazu Kapitel 4.3).

Der Austausch zwischen den genannten Modulen geschieht mittels einer Socket Verbindung, die nach dem Client-Server Prinzip aufgebaut ist. Welches Modul in diesem Fall die Rolle des Client und welches die Rolle des Servers übernimmt, ist bei nur einem Robotermodell nicht entscheidend, es bietet sich jedoch an, die »physikalische Umgebung« und Graphikausgabe als Server zu implementieren auf den dann mehrere Clients in Form von RCX Emulatoren zugreifen können.

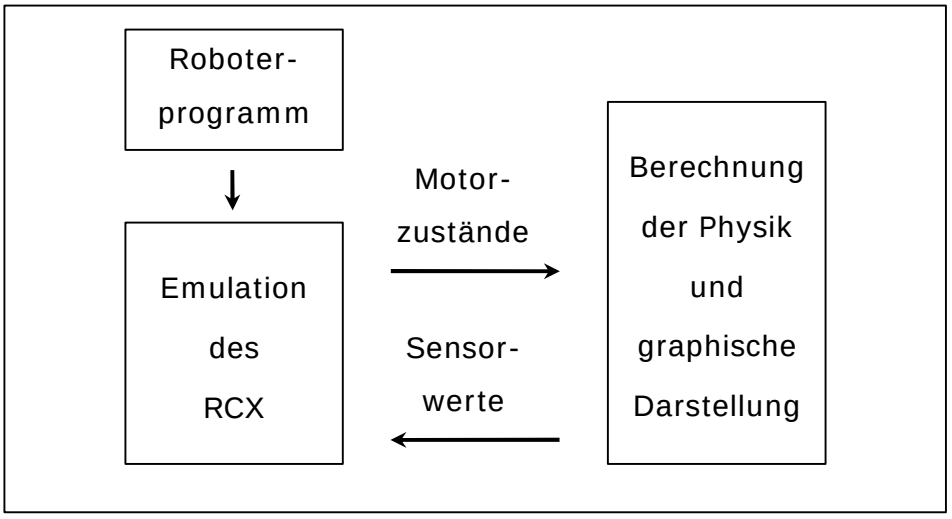

Abbildung 2: grundlegende Modularisierung des Produkts

4.2 Programmiersprachen und Werkzeuge

Die Auswahl der für den Zweck geeigneten Programmiersprachen wird teilweise durch die benutzten Entwicklungswerkzeuge bestimmt. In Abstimmung mit dem Projekt Roberta und dem Fraunhofer Institut AIS wird für die Berechnung der physikalischen

Abläufe das frei verfügbare Produkt *Open Dynamics Engine*[33] (ODE) eingesetzt, welches auch die Funktionalität der graphischen Ausgabe übernimmt (siehe Kapitel 4.3). Als Schnittstelle zur ODE ist die am Fraunhofer Institut AIS entwickelte *Open Dynamics Language*[34] (ODL) vorgesehen (siehe Kapitel 4.4), deren Schnittstelle auf C++ basiert. Die Benutzung dieses umfangreichen Werkzeugs ist daher auch nur mit einer Entwicklung in C++ sinnvoll.

Die Entwicklung des für die Nachbildung des RCX zuständigen Moduls unterliegt keinen solchen Vorgaben. Aus Überlegungen bezüglich der Entwicklungsgeschwindigkeit, der guten Dokumentierbarkeit, der Laufzeitstabilität und der Plattformunabhängigkeit resultiert die Entscheidung, diesen Teil des Produkts in Java™ zu programmieren. Nicht zuletzt soll auf diese Weise auch der Beweis angetreten werden, dass es durchaus möglich ist, Echtzeit-Simulationen in Java™ zu schreiben, obwohl von vielen Seiten behauptet wird, Java™ sei langsam, träge und ineffizient.

Die Anwendung besteht somit aus zwei getrennten Prozessen, von denen der eine in Java™ und der andere in C++ implementiert wird.

Die Java™ Komponente wird mit der Entwicklungsumgebung Eclipse entwickelt. Diese bietet eine komfortable Projektverwaltung.

Der auf C++ basierende Teil wird mit Microsoft Visual C++ entwickelt, wobei sowohl das Qt Plug-In (siehe Kapitel 4.6) als auch die ODE (siehe Kapitel 4.3) und die ODL (siehe Kapitel 4.4) verwendet werden.

---

[33] Die *Open Dynamics Engine* dient zur Simulation von physikalischen Vorgängen in frei konfigurierbaren Umgebungen. Benutzt wird diese Bibliothek über eine C-Schnittstelle.

[34] Die *Open Dynamics Language* ist eine Klassenbibliothek und dient zur einfacheren Benutzung der ODE.

4.3 Open Dynamics Engine (ODE)

Die Open Dynamics Engine (ODE) ist eine frei verfügbare Bibliothek, die von Russel Smith zur Simulation von bewegten Körpern in einer Umgebung entwickelt wurde. Die ODE eignet sich gut für die Simulation von Bodenfahrzeugen, Robotern mit Beinen und ganz allgemein sich bewegenden Objekten in virtuellen Umgebungen.

Die ODE wird im Rahmen des Mindstorms™ Simulators dazu benutzt, die Berechnung des Roboterverhaltens in einer Simulationsumgebung durchzuführen. Des weiteren stellt die ODE Funktionen zur graphischen Ausgabe des Roboters in seiner Umgebung bereit.

Die Entscheidung für die Benutzung der ODE resultiert in erster Linie daraus, dass am Fraunhofer Institut AIS die Entscheidung getroffen wurde, für Simulationsprojekte mit dieser Bibliothek zu arbeiten. Die Merkmale der ODE sind folgende:

- Die ODE ist für die Benutzung in interaktiven Echtzeit-Simulationen entwickelt worden. Sie eignet sich gut für die Simulation von sich bewegenden Objekten in veränderbaren Umgebungen.
- Die ODE ist schnell, robust, stabil und plattformunabhängig.
- Die ODE ist frei verfügbar und kann laut Lizenz auch in kommerziellen Produkten verwendet werde, ohne dass Gebühren fällig werden.
- Die ODE bietet eine Kollisionsüberprüfung, die schnell und genau arbeitet.
- Die ODE bietet Möglichkeiten zur graphischen Ausgabe einer Simulationsumgebung.

Die ODE wird im Rahmen dieser Arbeit über die Klassenbibliothek ODL benutzt, auf die in Kapitel 4.4 näher eingegangen wird. Ein englischsprachiges Benutzungshandbuch mit Referenzen und Internetquellen findet sich in Anhang C auf der CD-ROM.

4.4 Open Dynamics Language (ODL)

Die Open Dynamics Language (ODL) ist eine einfach zu benutzende Klassenbibliothek, die zur Erstellung von physikalischen Modellen in simulierten Umgebungen dient. Die ODL ist am Fraunhofer Institut AIS von Dr. Christian Verbeek entworfen worden und befindet sich noch in der Implementationsphase. Eine Projektbeschreibung findet sich im Internet unter: *http://ais.gmd.de/ROCK/ODL/index.en.htm*

Der Mindstorms™ Simulator ist eine der ersten Anwendungen, die Gebrauch von der ODL machen. Es steht keine Dokumentation der ODL zur Verfügung, sodass der Quelltext zunächst analysiert werden muss, um zu verstehen, wie die ODL benutzt werden kann. Zu diesem Zweck wurde zunächst ein Klassendiagramm der ODL Bibliothek erstellt, welches in Abbildung 3 dargestellt wird.

Die Klasse *Agent* steht hier beispielhaft für ein Robotermodell. Sie wird abgeleitet von der Klasse *IStructure*, wodurch sie eine Grundfunktionalität erhält. Sie erzeugt eine Instanz der Klasse *ICompound*, welche die einzelnen Elemente des Robotermodells in Form von Instanzen der Klasse *IObject* enthält.

Die ODE wird zur Entwicklung der Mindstorms™ Simulation eingesetzt, da sie eine objektorientierte Schnittstelle zur ODE bietet. Die Programmentwicklung ist in diesem Fall wesentlich einfacher und vor allem weniger fehleranfällig, als bei einer Verwendung der reinen C-Funktionen der ODE Schnittstelle.

Ist ein Robotermodell erst einmal erstellt, bietet die ODL Methoden für dessen Bewegung, den Einsatz von Sensoren und Motoren. Vor allem die Behandlung von Sensorereignissen wird durch die Benutzung der ODL wesentlich vereinfacht.

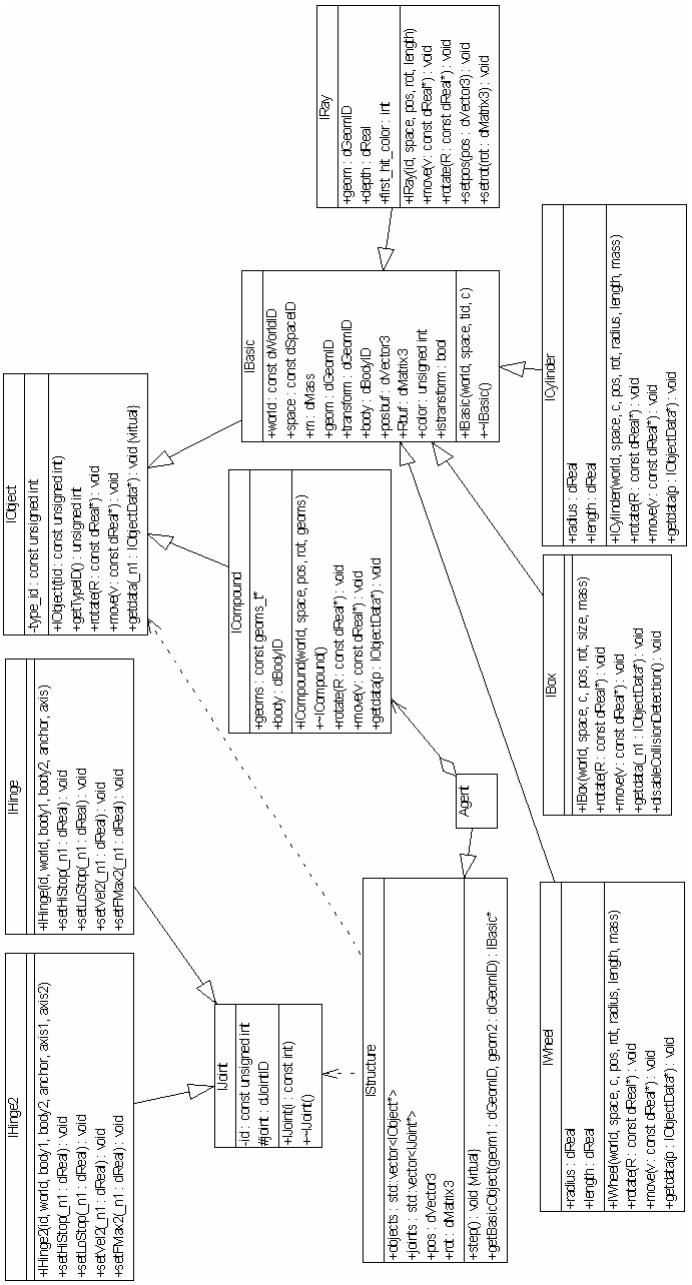

Abbildung 3: Klassendiagramm der Open Dynamics Language

4.5 Open Graphics Library (OpenGL)

OpenGL ist die im professionellen Bereich vorherrschende Programmierbibliothek für die Entwicklung von 2D und 3D Graphikprogrammen. Eine Reihe von Hardware- und Softwareherstellern beteiligten sich bei der Spezifizierung von OpenGL und unterstützen sie. Dadurch können OpenGL Programme relativ einfach auf praktisch jede Plattform portiert werden, sei es von einem Windows™ PC

nach Linux, auf eine High-End UNIX Workstation oder auch auf einen Mainframe. OpenGL ist von der Konzeption her unabhängig von Plattform oder Betriebssystem. Es ermöglicht Entwicklern, Programme zu schreiben, die relativ einfach auf vielen verschiedenen Plattformen eingesetzt werden können. Vor allem ist OpenGL eine effektive, leistungsfähige Graphikbibliothek, und es gibt viele Graphikkarten mit 2D und 3D Hardwarebeschleunigern, die OpenGL Funktionalitäten auf Hardwareniveau realisieren.

Für die Entwicklung der Simulation wird OpenGL hauptsächlich aus Gründen der Plattformunabhängigkeit eingesetzt. Da die ODE vorgefertigte Funktionen zur graphischen Ausgabe der Simulationsumgebung und der Robotermodelle in OpenGL bereitstellt, ist der Implementationsaufwand bei dessen Benutzung auch am geringsten.

4.6 Troll Tech Qt

Qt ist eine C++ Klassenbibliothek zur Entwicklung graphischer Benutzeroberflächen. Enwickelt wurde sie von der Firma Troll Tech AS in Oslo (http://www.troll.no/intro.html). Qt ist sehr schnell, kompakt und leicht zu erlernen. Das Hauptmerkmal von Qt ist die Möglichkeit, Programme mit graphischen Benutzungsoberflächen plattformunabhängig implementieren zu können. Für die Erstellung von Fenstern und anderen graphischen Elementen werden spezielle Qt Klassen benutzt, die sich für die verschiedensten Betriebssysteme kompilieren lassen.

Für die Entwicklung der Mindstorms™ Simulation wird Qt hauptsächlich eingesetzt, um das Produkt plattformunabhängig zu gestalten. Es ist somit nicht erforderlich für verschiedene Systeme wie Linux, Windows, MacOS eine eigene Fensterverwaltung zu programmieren. Außerdem stellt Qt auch Klassen zur plattformunabhängigen Implementation von Socket Verbindungen zur Verfügung, die im Rahmen der Simulation eingesetzt werden.

5 Emulation des RCX

Bei der Entwicklung einer Simulation für LEGO® Mindstorms™ Roboter besteht ein Großteil der zu bewältigenden Arbeit darin, die nachzubildenden Komponenten zu analysieren. Sowohl die internen Prozesse des Steuerungssystems als auch die physikalischen Eigenschaften eines typischen Robotermodells müssen genau betrachtet werden, um eine möglichst realitätsgetreue Modellbildung zu ermöglichen.

In diesem Kapitel werden die einzelnen Komponenten des Steuerungssystems (RCX) analysiert und erste Lösungsansätze in Form von Klassenentwürfen vorgestellt. Eine nähere Betrachtung des Roboterverhaltens und dessen Umsetzung folgt in Kapitel 6.

5.1 Das Dialogfenster des Emulators

Die Emulator-Anwendung ist auch für die Darstellung eines Dialogfensters zuständig, über das die Simulation und der Roboter gesteuert werden kann. Zu diesem Zweck wird eine Klasse *MainWindow* entworfen, die für die Erzeugung eines Anwendungsfensters und die Behandlung von Menüereignissen zuständig ist.

Die beim Programmstart erzeugte Instanz der Klasse *MainWindow* erzeugt eine Instanz der Klasse *RcxInstance*, die für die Behandlung der vier Bedienungsknöpfe zuständig ist. Sie ist die Repräsentation eines physikalischen RCX und stellt die Schnittstelle zwischen BenutzerIn und virtuellem RCX dar.

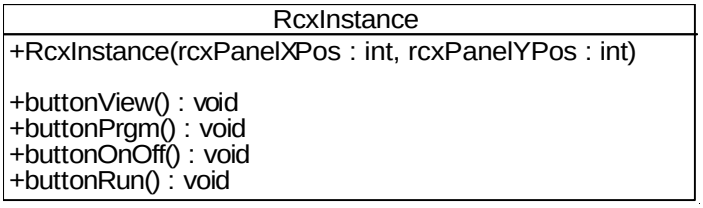

Abbildung 4: Klassenentwurf *RcxInstance*

Von *RcxInstance* aus werden Instanzen von *RcxVirtualMachine* (siehe dazu Kapitel 5.3) und *RcxPanel* erzeugt. Die Klasse *RcxPanel* ist für den graphischen Aufbau des virtuellen RCX und das Abfangen von Mausereignissen zuständig. Sie besitzt keine öffentlichen Schnittstellen, sondern ruft vielmehr die Methoden der Klasse *RcxInstance* auf, wenn z.B. eine der Bedienungstasten gedrückt wird.

Der graphische Aufbau des Displays wird getrennt in einer Instanz der Klasse *LcdDisplay* vorgenommen. Diese wird von *RcxPanel* erzeugt.

LcdDisplay
+LcdDisplay(xPos : int, yPos : int, width : int, height : int) +updateDisplay(lcdDisplayData : LcdDisplayData, forceRefresh : boolean) : void

Abbildung 5: Klassenentwurf *LcdDisplay*

Der jeweils aktuelle Zustand des Displays wird in einer Instanz der Klasse *LcdDisplayData* verwaltet.

LcdDisplayData
+LcdDisplayData() +LcdDisplayData(data : LcdDisplayData) +equals(data : LcdDisplayData) : boolean +setDisplayIntValue(value : int, zeroFill : boolean) : void

Abbildung 6: Klassenentwurf *LcdDisplayData*

5.2 RCX und Bytecode

Der RCX ist ein per Infrarotschnittstelle programmierbarer, mikroprozessor-basierter Baustein mit jeweils drei analogen Ein- und Ausgängen und vier Bedienungsknöpfen. Dieser Baustein ist das zentrale Element der RIS Baukästen und somit auch von jedem damit gebauten Roboter.

Die Rechenarbeit im RCX übernimmt ein Hitachi-H8 Prozessor, der mit 16 MHz getaktet ist. Er ist mit 16 Kilobyte ROM und 32 Kilobyte

RAM Speicher ausgestattet. Nach einem »Kaltstart« (z.B. nach dem Entfernen der Batterien für eine gewisse Zeit) steht nur das ROM zur Verfügung, welches im Prinzip nur eine rudimentäre Funktion zum Übertragen von Daten in das RAM zur Verfügung stellt. Programme oder Befehle können in diesem Zustand weder direkt ausgeführt, noch übertragen werden. LEGO® stellt zu diesem Zweck eine so genannte *Firmware*[35] bereit, die auch auf der mit dem RIS Baukasten mitgelieferten CD enthalten ist. Diese Firmware kann in das RAM des RCX geladen werden und ermöglicht dann das Übertragen und Ausführen von Befehlen und Programmen.

Die LEGO® Firmware erwartet Befehle im so genannten RCX Bytecode, d.h. sie enthält selbst eine *virtuelle Maschine*[36], welche die abzuarbeitenden Befehle Byte für Byte ausliest und in den eigentlichen Maschinencode des RCX übersetzt.

Um das Pflichtenheft zu erfüllen (siehe Kapitel 3.2.3), ist es für die Simulationsentwicklung nicht notwendig, den RCX komplett zu emulieren. Der Schwerpunkt liegt vielmehr auf einer Nachbildung der virtuellen Maschine (siehe Kapitel 5.3).

5.3 Virtuelle Maschine des RCX

Der Analyse der virtuellen Maschine des RCX liegt hauptsächlich die von LEGO® mit dem Mindstorms™ Software Development Kit (SDK)

---

[35] Mit *Firmware* wird der Betriebssystem-Kern für ein Mikroprozessorsystem bezeichnet. Während Betriebssysteme für größere Rechnersysteme meist auf einer Festplatte (Massenspeicher) gespeichert werden, und neben dem Kern noch viele andere Komponenten enthalten, wird in einem Mikroprozessorsystem meist nur der Kern benötigt. Wird dieser in einem Speicherbaustein (je nach System in RAM oder ROM) gespeichert, so wird er *Firmware* genannt.

[36] *Virtuelle Maschine* ist der Begriff für eine systemnahe Programmkomponente, welche Programme, die nicht aus Maschinencode bestehen, auf der jeweiligen Plattform ausführt. Das bekannteste Beispiel für eine *virtuelle Maschine* ist das Java Runtime Environment (JRE).

bereitgestellte *Command Overview*[37] zu Grunde, die als Quelle auf der beigefügten CD-ROM verfügbar ist (siehe Anhang D). Es wird darauf verzichtet im weiteren Verlauf dieses Kapitels auf einzelne Seiten dieses Dokuments zu verweisen.

Herzstück der virtuellen Maschine ist ein Bytecode Interpreter, der die Bytecode Kommandos ausführt. Für die Interpretation der Kommandos sind die im folgenden beschriebenen Eigenschaften und Zuständen der virtuellen Maschine relevant:

- 5 Programmspeicherplätze, von denen jeder bis zu 10 so genannte *Tasks*[38] und bis zu 8 Unterprogramme aufnehmen kann.
- 32 persistente, globale Variablen, auf die von allen Tasks und Unterprogrammen eines Speicherplatzes zugegriffen werden kann. Persistent bedeutet in diesem Zusammenhang, dass die Variablen nur bei einem Neustart des Systems auf null gesetzt werden.
- 16 lokale Variablen für jede Task, die unter anderem für eine sichere Parameterübergabe an Unterprogramme genutzt werden können. Lokale Variablen sind nur innerhalb der erzeugenden Task verfügbar.
- 4 Systemzeitgeber (engl.: timer), auf die von allen Tasks und Unterprogrammen zugegriffen werden kann. Die Zeitgeber laufen kontinuierlich von 0 bis 32767, von wo aus sie wieder bei 0 beginnen. Die Zeitgeber haben eine Auflösung von 100 Millisekunden.
- 3 Eingänge

---

[37] LEGO® Mindstorms™ RCX 2.0 Firmware Command Overview

[38] Mit *Tasks* werden parallel ablaufende Programmstränge bezeichnet, die der Programmierer selbst definieren, starten und anhalten kann. Eine nähere Beschreibung findet sich in Kapitel 5.3.3.

- 3 Ausgänge
- Eine Einheit zur Klangerzeugung

Diese Eigenschaften müssen von der Simulation ebenso nachgebildet werden, wie die Interpretation des Bytecodes selbst.

Zur Implementation:

Die Nachbildung der virtuellen Maschine ist das zentrale Element und somit auch die zentrale Klasse (*RcxVirtualMachine*) im Entwurf der RCX Emulation.

Sie enthält den Bytecode Interpreter (siehe Kapitel 5.3.1), der dafür verantwortlich ist, die Befehle abzuarbeiten. Die genannten Eigenschaften werden in Form von weiteren Klassen entwickelt, die für ihre eigene Funktionalität verantwortlich sind. Die Klasse der virtuellen Maschine fungiert gewissermaßen als leitende Einheit, die den Ablauf eines Mindstorms™ Programms durch gezielte Instanziierung und Benutzung der anderen Klassen steuert. Die folgende Abbildung zeigt den Klassenentwurf mit allen öffentlichen Methoden.

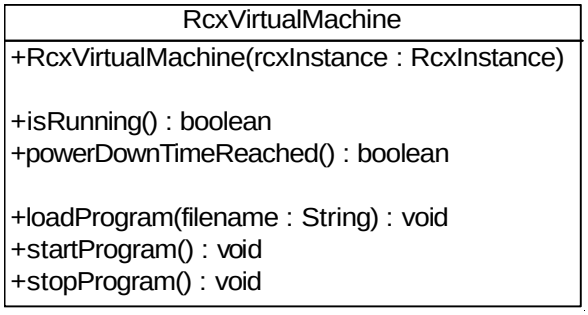

Abbildung 7: Klassenentwurf *RcxVirtualMachine*

5.3.1 RCX Bytecode Interpreter

Der RCX Bytecode Interpreter hat einen Befehlsumfang von insgesamt 85 ausführbaren Befehlen. Jeder dieser Befehle wird durch

einen ihm zugehörigen Bytewert, den so genannten *Opcode*[39], repräsentiert, wobei nicht alle möglichen Befehle für die Simulation relevant sind. Es werden nur solche Befehle implementiert, die für die eigentliche Programmausführung und die im Pflichtenheft festgehaltenen Mindestanforderungen benötigt werden. Nicht berücksichtigt werden Befehle, die den RCX direkt, also ohne Ablauf eines Programms, steuern. Ebenso wenig werden Befehle, die der Ansteuerung der Infrarotschnittstelle dienen, implementiert. Insgesamt müssen somit 64 Bytecode-Kommandos in der Simulation berücksichtigt werden.

Es gibt sowohl Befehle mit als auch Befehle ohne Parameter. Der Interpreter erkennt am Opcode die Anzahl und Art der folgenden Parameter. Eine Tabelle sämtlicher Befehle und deren Implementationsstatus findet sich in Anhang A.

Die für die Simulation relevanten Befehle lassen sich in verschiedene Gruppen einteilen:

- Befehle zur Programmablaufsteuerung (bedingte und unbedingte Sprungbefehle)
- Befehle zum Arbeiten mit Variablen
- Befehle zum Umgang mit Tasks
- Befehle zum Umgang mit Events
- Befehle zum Umgang mit Unterprogrammen
- Timerbefehle
- Zählerbefehle
- Klangerzeugungsbefehle

---

[39] Der Begriff *Opcode* stammt aus dem Englischen und ist eine Abkürzung für »Operation Code«. Er bezeichnet den Teil eines Assembler- oder Bytecodebefehls, der die Art der Befehlsausführung und die weitere Struktur des Befehls festlegt.

- Sensorbefehle
- Befehle zur Ansteuerung der Ausgänge
- sonstige Befehle

Zur Implementation:

Der Bytecode Interpreter selbst wird innerhalb der Klasse der virtuellen Maschine realisiert. Er hat die Aufgabe, jeweils ein Kommando zu verarbeiten, indem eine zugehörige Methode aufgerufen wird, welche für die weitere Bearbeitung zuständig ist.

Zum Zweck einer guten Lesbarkeit und Wartbarkeit des Quelltextes wird für jeden Opcode eine Konstante angelegt, auf deren Auftreten beim Programmablauf dann per »switch« Anweisung reagiert wird. Wird beim Anlegen der Konstanten sorgfältig gearbeitet, so sind weitere Tippfehler und Verwechslungen in der Abfrage so gut wie ausgeschlossen oder zumindest schnell auffindbar. Folgender Quelltextausschnitt verdeutlicht dieses Vorgehen:

```
public class RcxVirtualMachine extends Thread
{
   /*
    * static constants including all supported bytecodes
    */
   protected static final byte PB_ALIVE_OR_NOT= (byte) 0x10;
   protected static final byte STOP_ALL_TASKS=  (byte) 0x50;
   protected static final byte PB_TURN_OFF=     (byte) 0x60;
   protected static final byte CLEAR_SOUND=     (byte) 0x80;
   ...

   private int executeByteCommand(int taskIndex)
   {
      byte retval;
      ...
      byte command= program[program_slot][pc++];

      switch (command)
      {
         case PB_ALIVE_OR_NOT: retval= pbAliveOrNot(); break;
         case STOP_ALL_TASKS: retval= stopAllTasks(); break;
         case PB_TURN_OFF: retval= pbTurnOff(); break;
         case CLEAR_SOUND: retval= clearSound(); break;
```

```
        ...
      }
      return retval;
   }
   ...
}
```

5.3.2 Parameter Sources

Zu vielen Kommandos gehören Parameter in Form eines Paares von Source und Value. Die Source gibt den Typ des Parameters an und der Value ist typischerweise der Index dieses speziellen Parametertyps.

Folgendes Beispiel verdeutlicht diese Vorgehensweise:

Die Befehlsstruktur des Kommandos *SetVar* (Opcode 0x14) zum Setzen von Variablen wird in der Spezifikation wie folgt beschrieben:

0x14	Variable number	Source	Value (LO)	Value (HI)

Die »Variable number« gibt in diesem Fall an, welche der insgesamt 48 verfügbaren Variablen gesetzt werden soll. Die folgenden Einträge für Source und Value bestimmen den Wert, der der Variablen zugewiesen wird. Soll nun z.B. der Variablen Nummer 1 der Wert des Sensor an Eingang 2 zugewiesen werden, ergibt sich folgendes Bytecode Kommando: 0x14, 0x01, 0x09, 0x01, 0x00. Die Source 9 bezeichnet die Abfrage von Sensoren. Die darauf folgende 1 bezeichnet den Eingang 2 (es wird von 0 an gezählt).

Es sind insgesamt 26 Sources definiert, von denen bis auf eine Ausnahme alle für die Simulation implementiert werden müssen. Typische Sourcen sind z.B. Variablen, Sensorwerte, Motorzustände, Timerwerte und vieles mehr. Lediglich die Source mit der Bezeichnung »Message«, die für Kommandos der Infrarotschnittstelle zuständig ist, wird für die Umsetzung der virtuellen Maschine nicht benötigt. Eine Tabelle mit der (englischen) Beschreibung sämtlicher Sources findet sich in Anhang B.

Zur Implementation:

Zur Behandlung der Parameter wird die Klasse *RcxSources* bereitgestellt, die für die Verwaltung der Sources zuständig ist. Die Klasse stellt Methoden zum Abfragen und Setzen der Sources zur Verfügung.

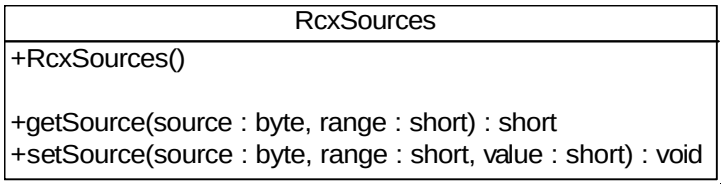

Abbildung 8: Klassenentwurf *RcxSources*

5.3.3 Tasks

Tasks sind Programmstränge, die vom Programmierer definiert, gestartet und angehalten werden können. Sie können parallel abgearbeitet werden. So kann z.B. eine Task für die Bewegung eines Roboters zuständig sein, während eine weitere Task Sensorberührungen abfängt und entsprechend darauf reagiert.

Zur Implementation

Zur Abarbeitung von Tasks wird eine Klasse *RcxTask* bereitgestellt. Für jede im Roboterprogramm erzeugte Task wird von der virtuellen Maschine eine Instanz dieser Klasse angelegt. Der Konstruktor erhält eine Referenz auf die virtuelle Maschine und den Einstiegspunkt der Task im Bytecode (program counter).

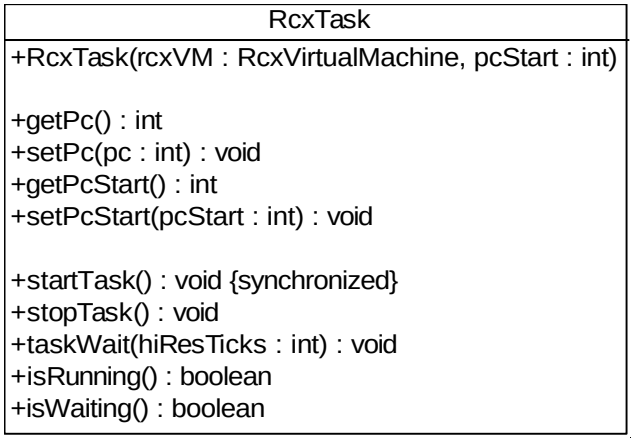

Abbildung 9: Klassenentwurf *RcxTask*

5.3.4 Events

Die Firmware erkennt und meldet im laufenden Programm bis zu 16 unabhängige Events sowohl für physikalische als auch virtuelle Sensoren. Die Meldung eines Events beeinflusst den Programmablauf von Tasks, die sich für eine Benachrichtigung von Events angemeldet haben.

Events können für folgende »Sensoren« generiert werden:

- physikalische Sensoren
- Systemzeitgeber
- Counter
- Infrarotnachrichten

Der komplette Umfang der möglichen Events (Eventtypen) ist folgender:

- Pressed Event – wird bei Betätigung eines Berührungssensors ausgelöst.
- Released Event – wird beim Loslassen eines Berührungssensor ausgelöst.
- Period Event – wird zunächst nicht implementiert

- Transition Event – wird zunächst nicht implementiert
- Change-rate exceeded event – wird zunächst nicht implementiert
- Enter low range – wird bei Absinken des Messwertes unter einen festgelegten Schwellwert ausgelöst.
- Enter normal range – wird ausgelöst, wenn der Messwert in einen nach unten und oben begrenzten Wertebreich eintritt.
- Enter high range – wird bei Ansteigen des Messwertes über einen festgelegten Schwellwert ausgelöst.
- Click – wird zunächst nicht implementiert
- Double click – wird zunächst nicht implementiert
- Mail box – wird zunächst nicht implementiert

Nach ausführlichen Testläufen konnte sichergestellt werden, dass die nicht implementierten Events bei keinem der in Frage kommenden Roboterprogramme auftreten.

Für jedes zu behandelnde Event müssen mehrere Attribute gesetzt werden, die für die Generierung der Events benötigt werden. Diese sind folgende:

- Upper Treshold – Oberer Schwellwert
- Lower Treshold – Unterer Schwellwert

Die beiden Schwellwerte legen die oben genannten Bereiche low, normal und high fest.

- Hysteresis – definiert, um wieviel die Schwellwerte überschritten werden müssen, bis ein Event ausgelöst wird.

Die virtuelle Maschine generiert bei Programmablauf Informationen zu den aufgetretenen Events. Die folgenden Daten können vom Roboterprogramm ausgelesen werden:

- Task events – eine Bitmaske, an der sich ablesen lässt, welche Events die aktuelle Task benachrichtigt haben
- Event state – der aktuelle Zustand eines Sensors: low, normal oder high

Zur Implementation

Die Klasse *RcxEvent* übernimmt die Behandlung von Events. Für jedes mögliche Event wird eine Instanz dieser Klasse erzeugt, wobei dem Konstruktor der gewünschte Sensor- und Eventtyp übergeben wird. Sie stellt neben einer Methode zur Kalibrierung von Events eine Methode *eventOccured()* bereit, über welche die übergeordnete Instanz abfragen kann, ob ein Event eingetreten ist.

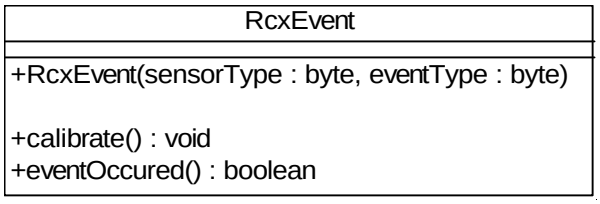

Abbildung 10: Klassenentwurf *RcxEvent*

5.3.5 Sensoren

Der RCX hat drei Eingänge, an die externe Sensoren angeschlossen werden können. Die Eingänge sind mit 1, 2 und 3 bezeichnet. Von LEGO® werden vier verschiedene Arten von Sensoren angeboten. Dies sind der Berührungssensor, der Lichtsensor, der Rotationssensor und der Temperatursensor. Generell können zwei Arten von Sensoren unterschieden werden: Es gibt zum einen Sensoren, die selber mit Strom versorgt werden müssen, um richtig zu arbeiten – diese werden »aktiv« genannt. Die andere Sensorvariante kommt ohne Stromversorgung aus und wird mit »passiv« bezeichnet. Diese jeweilige Eigenschaft eines Sensors nennt sich »Sensortyp« und muss zu Beginn eines Programms für jeden benutzten Eingang festgelegt werden, damit der RCX sinnvolle Werte auslesen kann.

Eine weitere Einstellung bei der Arbeit mit Sensoren ist der »Sensormodus«. Durch ihn wird bestimmt, wie die von einem Sensor gelieferten Werte zu interpretieren sind. Die Technik des RCX liefert nämlich zunächst einen so genannten »Rohwert«, der mit Hilfe eines 10-bit Analog/Digital-Wandlers erzeugt wird. Mit diesem Wert kann nur in den wenigsten Fällen sinnvoll gearbeitet werden, sodass meistens eine Umwandlung erforderlich ist. So werden Lichtwerte im Normalfall auf Werte zwischen 0 und 100 skaliert, bei einem Berührungssensor ist der ausgelesene Wert entweder 0 oder 1 (nicht gedrückt oder gedrückt).

Zur Implementation

Für die Behandlung der Sensoren wird eine Klasse *RcxSensor* realisiert, die Methoden zum Abfragen und Setzen des Sensortyps, des Sensormodus und der Sensorwerte bereitstellt.

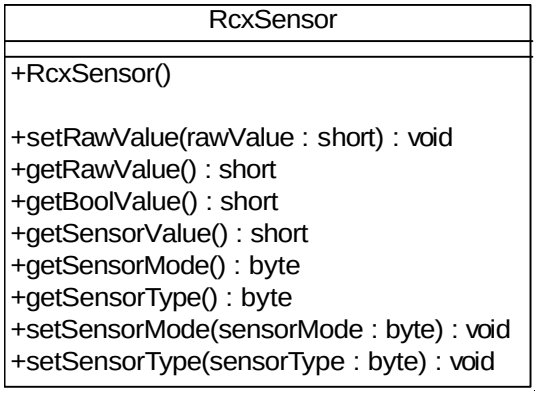

Abbildung 11: Klassenentwurf *RcxSensor*

5.3.6 Aktoren

Der RCX stellt drei Ausgänge zur Verfügung, über die externe Geräte, auch Aktoren genannt, mit Strom versorgt werden können. Diese Ausgänge sind mit A, B und C bezeichnet. Im Grunde genommen können die verschiedensten Arten von Aktoren verwendet werden, LEGO® selbst liefert im Rahmen des Mindstorms™-Programms aber nur zwei verschiedene: Lämpchen und Motoren. Da bei allen Anwendungen im Projekt Roberta ausschließlich Motoren verwendet

werden, werden auch nur diese im Rahmen der Simulation behandelt.

Beim Ansteuern eines Motors muss der Betriebsmodus angegeben werden. Die möglichen Betriebsmodi sind: Vorwärtslauf, Rückwärtslauf, Leerlauf und Bremsen. Des Weiteren kann die Ausgangsleistung eingestellt werden.

Zur Implementation

Für die Ansteuerung der Motoren wird eine Klasse *RcxMotor* realisiert, die Methoden zum Abfragen und Setzen der Motorzustände bereitstellt.

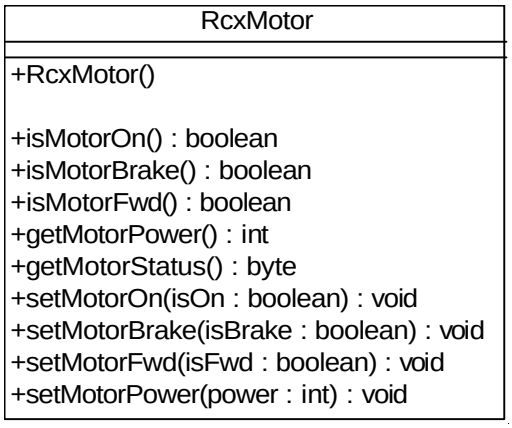

Abbildung 12: Klassenentwurf *RcxMotor*

5.3.7 Klangerzeugung

Der RCX verfügt über einen eingebauten Lautsprecher, über den er bestimmte Systemklänge und sogar ganze Tonfolgen ausgeben kann. Dies kann sehr nützlich sein, wenn ein Roboter eine Informationen über seinen Zustand ausgeben soll, ohne dass dabei ständig auf das Display geschaut werden muss. Es kann aber auch einfach nur lustig sein, wenn ein Roverbot umherfährt und dabei Musik spielt.

Zur Implementation

Für die Realisierung der Klangeinheit wird eine Klasse *RcxSound* realisiert, die Methoden zum Abspielen von Systemklängen und Tönen bereitstellt.

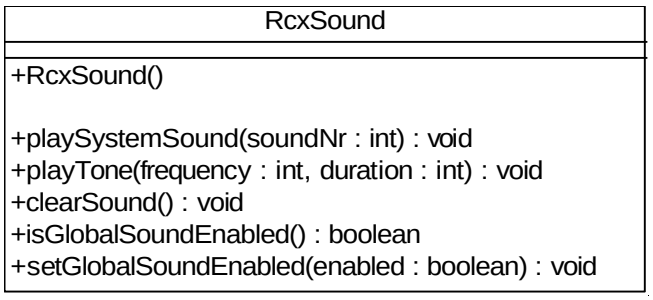

Abbildung 13: Klassenentwurf *RcxSound*

5.3.8 Timer

Bei der Programmierung von eingebetteten Systemen spielt die Messung der Zeit oft eine sehr große Rolle. Egal ob Motoren für einen bestimmten Zeitraum laufen sollen, Sensorwerte in einem festgelegten Intervall abgefragt werden sollen oder Daten für eine bestimmte Zeit gepuffert werden müssen: Bei all diesen Aufgaben (und vielen mehr) benötigt man eine Möglichkeit, die Zeit bzw. die Zeitspanne zwischen zwei Ereignissen zu messen.

Die LEGO®-Firmware stellt zu diesem Zweck vier getrennte Zeitzähler (engl.: timer) bereit. Sie lassen sich mit einer Genauigkeit von bis zu 10 Millisekunden auslesen.

Zur Implementation

Die Klasse *RcxTimer* stellt Methoden für das Setzen und Abfragen der Zeitzähler zur Verfügung. Neben den für die Roboterpramm-Ausführung nötigen Methoden *getTicks* und *getHiResTicks* enthält sie noch weitere, für den internen Simulationsablauf benötigte Methoden.

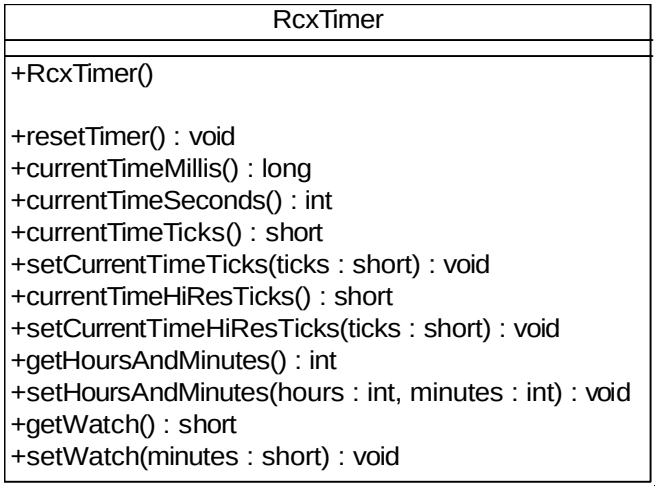

Abbildung 14: Klassenentwurf *RcxTimer*

5.3.9 Variablen und Counter

Die Benutzung von globalen Variablen wird in der Klasse der virtuellen Maschine implementiert. Die 32 verfügbaren Variablen werden dort angelegt und verwaltet, verschiedene Tasks können hier darauf zugreifen. Lokale Variable werden in der Klasse RcxTask realisiert.

Die drei verfügbaren Counter sind mit den ersten drei globalen Variablen identisch. Der Unterschied besteht darin, dass sie für die Eventgenerierung verwendet werden können. Die Counter werden somit, genau wie die Variablen, in der Klasse der virtuellen Maschine implementiert.

5.4 Kommunikation mit dem Server

Der RCX Emulator versucht beim Programmstart eine Socket Verbindung mit dem Simulationsserver herzustellen, der für die Berechnung des Roboterverhaltens und dessen graphischer Ausgabe zuständig ist. Diese Verbindung dient dem Austausch von Motorzuständen, Sensorwerten und Befehlen. In jedem Simulationsschritt werden die aktuellen Motorzustände an den Simulationsserver gesendet und die dort gemessenen Sensordaten empfangen. Auch Anweisungen wie »Neues Robotermodell laden« werden vom Emulator aus per Socket Verbindung an den Server geschickt.

Zu diesem Zweck wird eine Klasse SimEnvironment bereitgestellt, die Methoden zur Durchführung eines Simulationsschritts und zur Übermittlung von Kommandos an den Server enthält.

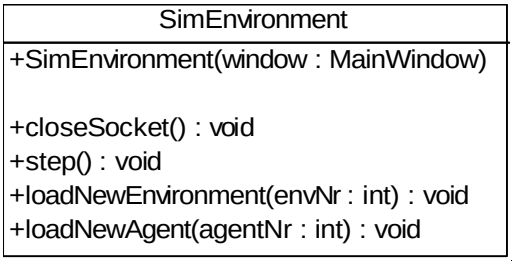

Abbildung 15: Klassenentwurf *SimEnvironment*

5.5 Programmiermöglichkeiten

Es gibt viele verschiedene Möglichkeiten, die Mindstorms™ Roboter zu programmieren. Im Rahmen dieser Arbeit ist nur die Programmierung mit RIS und NQC von Interesse. Dieser Abschnitt behandelt die Art und Weise der Roboterprogrammierung mit diesen beiden Compilern.

5.5.1 Robotics Invention System (RIS)

Die mit Robotics Invention System (RIS) bezeichneten Baukästen der Mindstorms™ Serie beinhalten eine gleichnamige Software, die zur Erstellung der Roboterprogramme verwendet werden kann. Programme werden hierbei in graphischer Form erstellt. Die verschiedenen Befehle und Kontrollstrukturen werden in Form von bausteinartigen Elementen mit der Maus zu einem Gesamtgefüge zusammengesetzt. Die folgende Abbildung zeigt ein Beispielprogramm:

Abbildung 16: Ein RIS Beispielprogramm

Der linke Programmstrang wird beim Programmstart ausgeführt. Er veranlasst den Roboter mittels einer Endlosschleife, immer weiter umher zu fahren. Das Element rechts mit der Beschriftung »Wenn Betätigt«, ist ein so genannter »Sensor Watcher«. Ist ein solches Element vorhanden, wird das in ihm festgelegte Sensorereignis bei der Programmausführung ständig abgefragt. In diesem Fall ist als Ereignis die Betätigung eines Berührungssensor an Eingang 2 festgelegt. Wird ein solcher Sensor nun gedrückt, wird der laufende Programmstrang unterbrochen und die Programmausführung wird bei dem Element, welches sich unter dem Sensor Watcher befindet, fortgesetzt. In diesem Beispiel wird eine Ausweichbewegung des Roboters durchgeführt. Nach Abarbeitung der Elemente des Sensor Watchers wird das Programm wieder im Hauptstrang fortgeführt.

Die Benutzung des der RIS Software ist leicht zu erlernen und bietet Menschen ohne Programmiererfahrung einen guten Einstieg in die Roboterprogrammierung.

5.5.2 Not Quite C (NQC)

Not Quite C (NQC) ist eine alternative Programmiersprache für Mindstorms™ Roboter. Sie wurde von Dave Baum entwickelt und ermöglicht die Programmierung der Roboter in einer C-ähnlichen Sprache. Der Compiler und die dazugehörige Entwicklungsumgebung »Bricx Command Center« sind frei verfügbar.

Um das gleiche Roboterverhalten zu erreichen, wie es durch das RIS Programm aus Abbildung 16 festgelegt wird, kann in NQC z.B. folgendes Programm geschrieben werden:

```
#define PRESSED 1

task main()
{
   SetSensor(SENSOR_2, SENSOR_TOUCH);
   start sensor_watcher;

   while(true)
   {
      OnFwd(OUT_A+OUT_C);
      Wait(100);
      Off(OUT_A);
      Wait(50);
      On(OUT_A);
      Wait(100);
      OnRev(OUT_C);
      Wait(70);
   }
}

task sensor_watcher()
{
   while(true)
   {
      if(SENSOR_2 == PRESSED)
      {
         stop main;
```

```
        OnRev(OUT_A+OUT_C);
        Wait(100);
        OnFwd(OUT_C);
        Wait(150);
        start main;
      }
   }
}
```

Der auffallendste Unterschied zu der RIS Programmierung liegt hier in folgenden Punkten:

- Die Motoren müssen einzeln angesteuert werden – es gibt keine Befehle wie »Vorwärts« oder »Rechtskehre«. Ebenso ist es nicht möglich Anweisungen wie »Fahre für eine Sekunde« zu geben – es werden vielmehr Motorzustände gesetzt und danach die gewünschte Zeit gewartet. Nach der Wartezeit ist das Programm für eine erneute Einstellung der Motoren verantwortlich.

- Die parallel ablaufenden Programmstränge der Roboterbewegung und der Sensorüberwachung werden hier in Form von »Tasks« angelegt, die vom Programm gezielt gestartet und angehalten werden müssen. Die Task »main« wird bei Programmstart ausgeführt.

NQC ist im Bereich der Lehre wahrscheinlich die am häufigsten benutzte Programmiersprache für Mindstorms™ Roboter. Es ist gut zu erlernen und bietet über die Roboterprogrammierung einen spielerischen Einstieg in Hochsprachen wie C.

5.6 Bytecode-Erzeugung

Sowohl die RIS Umgebung, als auch der NQC Compiler dienen dem Zweck, aus den in einer höheren Sprache geschriebenen Roboterprogrammen, RCX Bytecode zu erzeugen und über den Infrarot-Transmitter auf den Roboter zu übertragen. Die folgende Abbildung verdeutlicht diese Vorgehensweise:

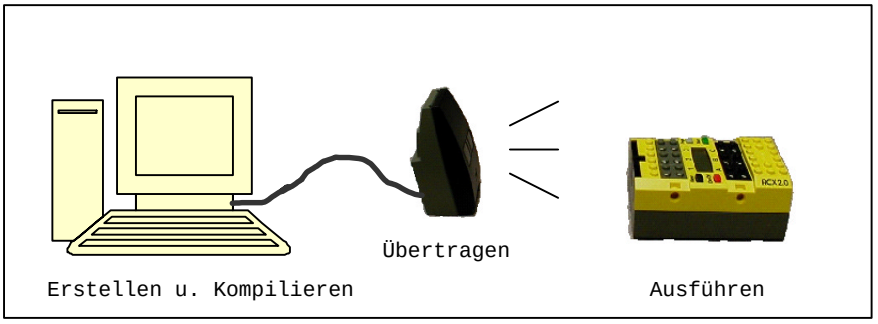

Abbildung 17: Übertragung der Programme beim realen RCX

Der zunächst naheliegende Weg, den Bytecode, der normalerweise über den Transmitter übertragen wird, genau in der Form in die Simulation zu importieren, birgt einen für den Rahmen dieser Arbeit nicht machbaren Implementationsaufwand. Dies liegt daran, dass die seriellen Schnittstellen eines PCs (dabei spielt es keine Rolle, ob COM oder USB) immer mit einem exklusiven Zugriff angesteuert werden. Das bedeutet, dass die Simulationsanwendung nicht ohne größeren Aufwand (Treiberprogrammierung, etc.) auf die dort gesendeten Daten zugreifen kann.

Der für die Implementation gewählte Weg besteht aus dem Import der abgespeicherten NQC und RIS Programme in die Simulation. Die Programme werden von der Simulation in den auszuführenden Bytecode übersetzt.

5.6.1 RIS Programme

Die RIS Programmierumgebung speichert Programme im so genannten »LEGO MindScript«. Die abgespeicherten Dateien haben typischerweise die Endung **.lsc*. In den MindScript Dateien ist der Programmablauf so gespeichert, dass er auch in der RIS Software graphisch wiedergegeben werden kann.

Das in Abbildung 16 gezeigte Programm sieht in abgespeicherter Form folgendermaßen aus:

```
program test {

   #include <RCX2.h>
```

```
   #include <RCX2MLT.h>
   #include <RCX2Sounds.h>
   #include <RCX2Def.h>
   sensor touch2 on 2
   touch2 is switch as boolean
   event tPress_touch2EventPress when touch2.pressed

   main {
      ext InterfaceType "kFreestyle"
      rcx_ClearTimers
      bbs_GlobalReset([A B C])
      start TouchWatcher0
      rcx_Priority( 8)
      trigger tPress_touch2EventPress
      try {
         forever {
            bb_Forward(A, C, 100)
            bb_TurnRight(A, C, 50)
            bb_Forward(A, C, 100)
            bb_SpinLeft(A, C, 70)
         }
      } retry on fail
   }

   watcher TouchWatcher0 monitor tPress_touch2EventPress
   {
      rcx_Priority( 3 )
      try {
         bb_Backward(A, C, 100)
         bb_SpinLeft(A, C, 150)
      } restart on fail
   } restart on event
}
```

Die Erzeugung des Bytecodes aus dieser Datei ist mit verfügbaren Tools nicht direkt möglich. Das LEGO® Mindstorms™ Software Developer Kit (SDK) bietet Möglichkeiten zur Bytecode-Erzeugung, die von dem im »Bricx Command Center« enthaltenen MindScript-Compiler *lcc32.exe* genutzt werden. Der von diesem Compiler mit Hilfe der vom Mindstorms™ SDK eingerichteten Bibliotheken erzeugte Bytecode kann jedoch nicht in Dateiform abgespeichert werden, sondern lässt sich nur direkt an den Infrarot-Transmitter übertragen.

Möglich ist jedoch die Erzeugung einer Datei mit so genanntem LEGO® Assembler Code (LASM-Code). Das oben abgedruckte Programmbeispiel sieht in LASM-Code folgendermaßen aus:

```
;Program test
;--------------------------------------
   task  0
Main:
   sent  0,0
   senm  0,0,0
   sent  1,0
   senm  1,0,0
   sent  2,0
   senm  2,0,0
   setv  31,2,0
   sent  1,1
   senm  1,1,0
   sete  0,1,0
   set   29,0,2,0
   set   28,0,2,0
   tmrz  0
   tmrz  1
   tmrz  2
   out   1,7
   dir   2,7
   pwr   7,2,7
   gout  2,7
   gdir  2,7
   gpwr  7,2,7
   speak
   msgz
   set   4,0,26,0
   start 1
   setp  7
;IfTrigger001:
   chkl  2,1,2,9,1,EndIfTrigger0001
   event 2,1
EndIfTrigger0001:
Retry0002:
   monal 9,Retry0002
Forever0003:
   dir   2,5
   out   2,5
   wait  2,100
   out   0,5
   dir   2,5
   out   1,4
```

```
   out   2,1
   wait  2,50
   out   0,5
   dir   2,5
   out   2,5
   wait  2,100
   out   0,5
   dir   2,4
   dir   0,1
   out   2,5
   wait  2,70
   out   0,5
   jmp   Forever0003
   monax
EndMain:
   endt
;---------------------------------------
;Main (task 0) code size: 142 bytes
;---------------------------------------
;Watcher TouchWatcher0
   task  1
Watch0001:
   mone  2,1,EventCheck0001
EventWait0004:
   jmp   EventWait0004
EventCheck0001:
   setv  47,23,1
   mone  2,1,EventCheck0001
   setp  2
   monal 9,Watch0001
   dir   0,5
   out   2,5
   wait  2,100
   out   0,5
   dir   2,4
   dir   0,1
   out   2,5
   wait  2,150
   out   0,5
   monax
   jmp   Watch0001
Restart0001:
   jmp   EventCheck0001
   endt
;---------------------------------------
;Watcher TouchWatcher0 (task 1) code size: 50 bytes
;---------------------------------------
;Total code size: 192 bytes
```

Zur Implementierung

Der frei verfügbare MindScript Compiler *lcc32* wird eingesetzt, um die in der RIS Software abgespeicherten Programme zunächst in LASM Code zu übersetzen. Diese Datei wird dann vom Emulator geladen und in den eigentlichen Bytecode übersetzt. Zu diesem Zweck wird eine Klasse *BytecodeTranslator* bereitgestellt, die den LEGO® Assembler Code in den eigentlichen Bytecode übersetzt.

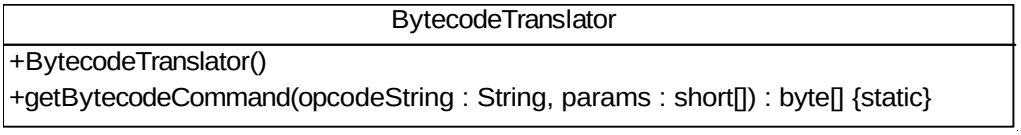

Abbildung 18: Klassenentwurf BytecodeTranslator

Die Methode *getBytecodeCommand* übersetzt jeweils eine Zeile des LASM Codes in das entsprechende Bytecode Kommando. Beim Import einer MindScript Datei wird diese zunächst per *lcc32* in eine temporäre LASM Datei umgewandelt und diese Datei dann über die Klasse *ByteCodeTranslator* eingelesen.

5.6.2 NQC Programme

Für den Import von NQC Programmen kann der frei verfügbare NQC Compiler benutzt werden, um eine so genannte NQC-Listing Datei zu erzeugen, die auch den benötigten Bytecode enthält. Zu diesem Zweck muss der Compiler mit einem speziellem Switch aufgerufen werden, der wiederum die Erzeugung der Listing Datei veranlasst.

Das in Abschnitt 5.5.2 abgedruckte Programmbeispiel wird vom Compiler in folgende Listing Datei übersetzt, die jweils rechts in einer Zeile den übersetzten Bytecode enthält:

```
*** Task 0 = main
000 pwr         ABC, 7                13 07 02 07
004 dir         ABC, Fwd              e1 87
006 sent        1, Switch             32 01 01
009 senm        1, Boolean            42 01 20
012 start       1                     71 01
014 dir         AC, Fwd               e1 85
```

```
016 out         AC, On                  21 85
018 wait        100                     43 02 64 00
022 out         A, Off                  21 41
024 wait        50                      43 02 32 00
028 out         A, On                   21 81
030 wait        100                     43 02 64 00
034 dir         C, Rev                  e1 04
036 out         C, On                   21 84
038 wait        70                      43 02 46 00
042 jmp         14                      27 9d

*** Task 1 = sensor_watcher
000 chk         1 != Input(1), 27       85 82 09 01 00 01 15
007 stop        0                       81 00
009 dir         AC, Rev                 e1 05
011 out         AC, On                  21 85
013 wait        100                     43 02 64 00
017 dir         C, Fwd                  e1 84
019 out         C, On                   21 84
021 wait        150                     43 02 96 00
025 start       0                       71 00
027 jmp         0                       27 9c

Total size: 73 bytes
```

Zur Implementierung

Der frei verfügbare Compiler *nqc* wird eingesetzt, um die in NQC vorliegenden Programme zunächst in eine NQC Listing Datei zu übersetzen. Diese Datei wird dann vom Emulator geladen und der in ihr enthaltene Bytecode extrahiert. Zu diesem Zweck wird eine Klasse *BytecodeSupplier* bereitgestellt, die den eigentlichen Bytecode in die Simulation importiert.

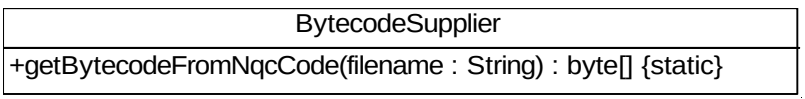

Abbildung 19: Klassenentwurf BytecodeSupplier

Die Methode *getBytecodeFromNqcCode* extrahiert den kompletten Bytecode aus einem NQC Listing und stellt diesen für den Emulator bereit. Beim Import einer NQC Datei wird diese zunächst per *nqc* in

eine temporäre Listing Datei umgewandelt und diese Datei dann über die Klasse *ByteCodeSupplier* eingelesen.

5.7 Klassendiagramm des Emulators

Der Gesamtentwurf des Emulators lässt sich am besten in Form eines Klassendiagramms darstellen:

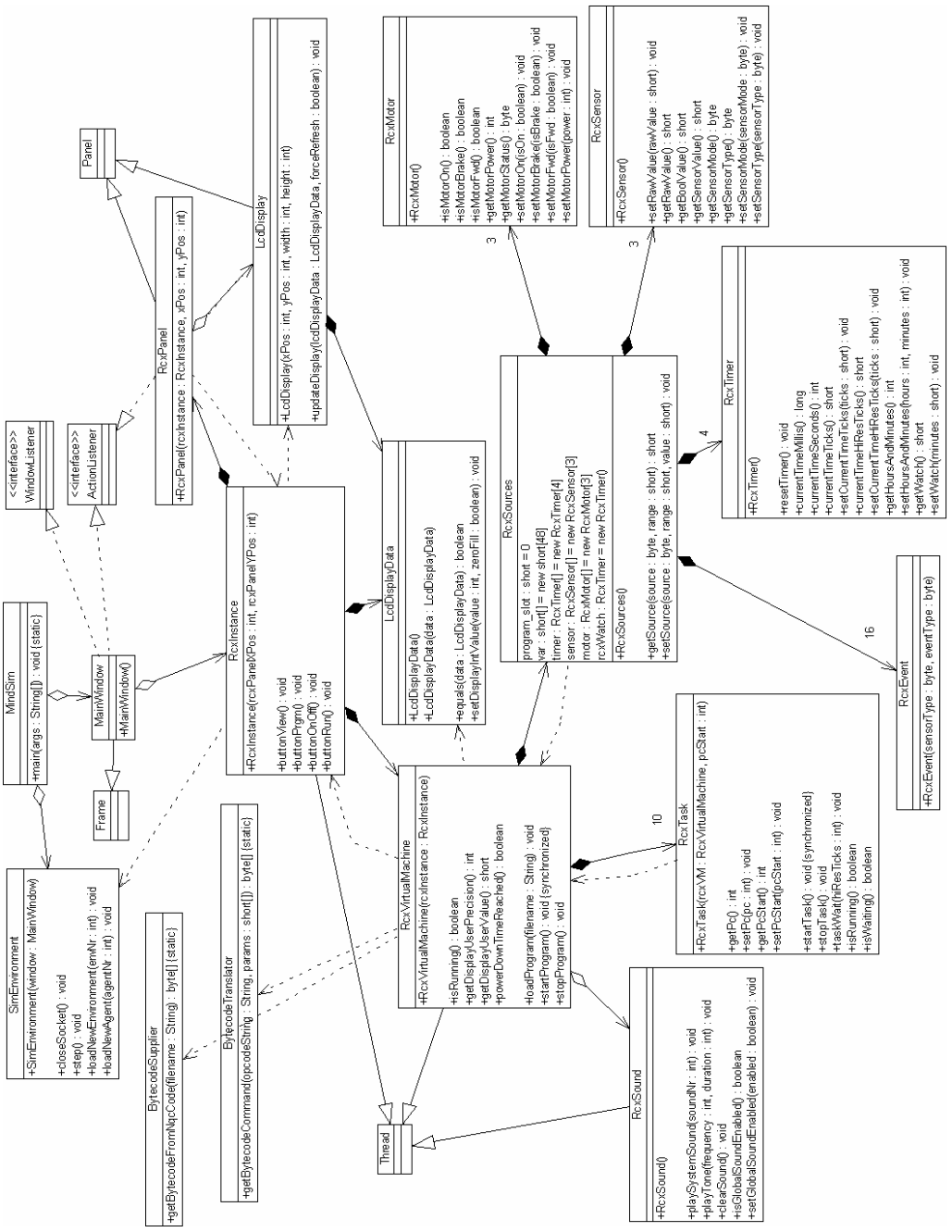

Abbildung 20: Klassendiagramm des Emulators

6 Berechnung und Darstellung des Roboterverhaltens

Zur Berechnung und Darstellung des Roboterverhaltens wird eine eigene Anwendung bereitgestellt. Diese Anwendung ist dafür zuständig, das physikalische Verhalten eines Robotermodells in Form von Simulationsschritten zu berechnen und die jeweils aktuelle Konstellation von Robotermodell und Umgebung graphisch zu präsentieren.

Die Anwendung macht intensiven Gebrauch von der Open Dynamics Engine (ODE) und der dafür verfügbaren Klassenbibliothek Open Dynamics Language (ODL). Eine nähere Beschreibung dieser Tools findet sich in Kapitel 4.3 und 4.4.

Des weiteren wird Qt verwendet, eine Bibliothek zur Unterstützung der Entwicklung plattformunabhängiger Anwendungen. Eine nähere Beschreibung findet sich in Kapitel 4.6.

6.1 Die graphische Ausgabe

Zur Erzeugung des Hauptfensters wird eine Klasse *MindSimDialog* entwickelt, die über die Klasse *MindSimDialogBase* von der Qt-Klasse *QDialog* abgeleitet ist. Sie erhält von dort die Funktionalität, auf Mausereignisse, Veränderungen der Fenstergröße und die Anforderung, dass Fenster zu schließen, zu reagieren.

Die Klasse *MindSimDialog* erzeugt eine Instanz der Klasse *GLView*, die für die graphische Ausgabe der Simulationsumgebung und des Robotermodells zuständig ist.

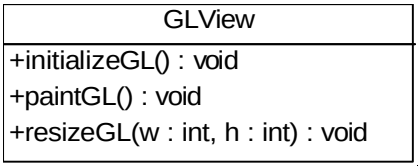

Abbildung 21: Klassenentwurf *GLView*

Die Klasse GLView ist von der Qt-Klasse QGLWidget abgeleitet, wodurch sie die Funktionalität erhält, dreidimensionale Welten per OpenGL auszugeben (zu OpenGL siehe Kapitel 4.5).

6.2 Das Robotermodell

Obwohl mit den Bauteilen der LEGO® Mindstorms™ Serie jede Menge verschiedener Konstruktionen gebaut werden können, bezieht sich ein Großteil der verfügbaren Beispiele und Bauanleitungen auf einen kleinen, fahrbaren Roboter. Der Grundgedanke bei dessen Konstruktion ist einfach: Ein RCX wird mit einem Chassis verbunden, an dessen Seiten vier Räder so angebracht sind, wie bei einem PKW. Es werden zwei Motoren verwendet, um jeweils die linken und die rechten Räder anzutreiben. Vorne am Chassis befindet sich eine Vorrichtung, an der Sensoren befestigt werden können.

Es sind im viele ähnliche solcher Konstruktionen denkbar. LEGO® bietet in der mit dem RIS Baukasten mitgelieferten Constructopedia™ eine beispielhafte Bauanleitung für einen solchen Roboter an. Der Grundaufbau wird von LEGO® »Roverbot« genannt, wobei mehrere verschiedene Antriebsarten möglich sind (Raupenketten, Beine, Räder...).

Es ist egal, wie die Motordrehung in eine Bewegung des Roverbots umgesetzt wird. Entscheidend ist, wie der Roverbot zu bestimmten Bewegungen veranlasst wird. Dies ist sehr einfach: Zum Vorwärtslaufen müssen beide Motoren auf Vorwärtslauf gestellt werden, zum Rückwärtslaufen dem entsprechend auf Rückwärtslauf. Für das Fahren einer Rechtskurve, wird nur der linke Motor angetrieben, für eine Linkskurve nur der rechte. Wenn sich beide Motoren entgegengesetzt drehen, dreht sich der Roverbot auf der Stelle.

Die folgende Abbildung zeigt das typische Grundmodell des Roverbot.

Abbildung 22 - Der Roverbot, ein fahrbarer Roboter

Für die Abbildung in der Simulation müssen die Grundstrukturen des Roverbot exakt vermessen werden. Relevant für eine realitätsgetreue Nachbildung sind die folgenden Komponenten:

- Das untere Chassis, welches auch die Motoren aufnimmt
- Die Räder und Zahnräder
- Der RCX, bestehend aus einer grauen und einer gelben Hälfte
- Die Infrarotschnittstelle (zur besseren Unterscheidung von vorne und hinten)
- Die vorne und hinten herausstehenden Bauteile, an denen Sensoren befestigt werden können

Zur Implementation

Die Hauptarbeit bei der Implementation eines Robotermodells per ODL liegt in der Bestimmung der Grundkomponenten eines solchen Modells, deren Vermessung und Umsetzung mit geeigneten von der ODE zur Verfügung gestellten physikalischen Objekten.

Die Maße des Roverbot müssen so skaliert werden, dass bei der Programmierung möglichst Werte zwischen 0,01 und 10 benutzt werden. Dies hängt mit den Algorithmen der Open Dynamics Engine

(Siehe Kapitel 4.3) zusammen, die bei niedrigeren oder höheren Werten starke Rundungsfehler aufweisen. Die Gesamtlänge des Roverbot beträgt ca. 26 Zentimeter (ohne Sensoren), wobei die kleinste zu berücksichtigende Abmessung 0,6 Zentimer lang ist. Es liegt nahe, hier zur Skalierung den Faktor 10 zu verwenden. Zusammen mit der Tatsache, dass in der ODE nie die Gesamtlänge eines Objekts, sondern nur die Strecke von dessen Mittelpunkt bis zum Rand, angegeben werden muss (was noch einmal den Faktor 2 ausmacht), ergeben sich für die ODE handhabbare Werte zwischen 0,03 und 1,3.

Des weiteren müssen sich sämtliche Maßangaben auf den Schwerpunkt des Roboters beziehen. Um den Aufwand nicht zu groß werden zu lassen, wird für die Implementation ein idealisierter Roverbot umgesetzt, dessen unteres Chassis das gesamte Gewicht ausmacht. Alle anderen Bauteile erhalten ein Gewicht von null. Legt man nun diesen Massenschwerpunkt fiktiv in den Ursprung eines dreidimensionalen, kartesischen Koordinatensystems, so lassen sich die benötigten Positionen und Bemaßungen der weiteren Komponenten recht gut ablesen.

Zur Implementierung

Zum Zweck der Realisierung wird eine Klasse *Agent* bereitgestellt, die ein Robotermodell im Sinne der ODL repräsentiert. Sie stellt Methoden zur Durchführung eines Simulationsschritts sowie zum Setzen der Motorzustände und Abfragen der Sensorwerte bereit.

Die Klasse *Agent* ist abgeleitet von *IStructure*, der grundlegenden Klasse aller Elemente in der ODL.

Agent
+Agent(parent : Sim*, w : dWorldID, s : dSpaceID) +~Agent() +step() : void +getSensorValue1() : int +getSensorValue2() : int +getSensorValue3() : int +setMotors(motor_status : unsigned char*) : void +getSensorValues(values : char*) : void

Abbildung 23: Klassenentwurf *Agent*

6.3 Die Simulationsumgebung

Ein Robotermodell braucht eine Umgebung, in der es sich bewegen und agieren kann. Die ODL bietet die Möglichkeit, solche Umgebungen aus verschiedenen Grundobjekten wie Quader, Kugel, etc. zu erstellen. Es wird eine Klasse *Environment* bereitgestellt, die bei Instanziierung eine von verschiedenen vorgegeben Umgebungen einrichtet. Ist eine Umgebung einmal eingerichtet, so bedarf sie keiner weiteren Funktionalität. Sämtliche physikalischen Ereignisse, die in dieser Umgebung von Interesse sein könnten, können über die ODL abgefragt werden.

Environment
+Environment(world : dWorldID, space : dSpaceID, env_nr : int) +~Environment()

Abbildung 24: Klassenentwurf *Environment*

Die Klasse *Environment* ist abgeleitet von *IStructure*, der grundlegenden Klasse aller Elemente in der ODL.

6.4 Die Simulation des physikalischen Verhaltens

Die Einrichtung der Simulation des physikalischen Verhaltens erfordert folgende Schritte:

- Einrichtung einer virtuellen Welt mit physikalischen Konstanten wie Gravitation, etc.

- Erzeugung einer Simulationsumgebung
- Erzeugung eines Robotermodells

Während der Ausführung eines Simulationsschritts müssen die folgenden Aktionen durchgeführt werden:

- Empfangen der Motorzustände vom Emulator (Client)
- Setzen der Motoren des Robotermodells auf die aktuellen Zustände
- Sämtliche in der Simulation vorhandenen Elemente vollziehen einen Simulationsschritt.
- Abfrage und Behandlung von physikalischen Kollisionen
- Ermitteln der Sensorwerte
- Senden der Sensorwerte an den Emulator (Client)
- graphische Ausgabe des aktuellen Zustands der Umgebung und des Roboters

Zur Implementation:

Zum Zwecke der Einrichtung der Simulation und zur Durchführung und Steuerung der Simulationsschritte wird eine Klasse *Sim* entwickelt, welche bei Instanziierung die Einrichtung vornimmt und eine Methode zur Durchführung eines Simulationsschritts bereitstellt.

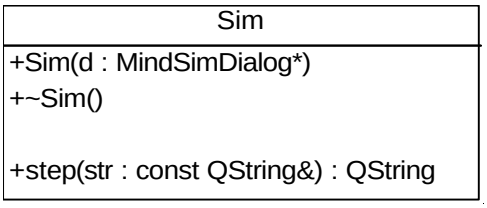

Abbildung 25: Klassenentwurf *Sim*

6.5 Kommunikation mit dem Client

Die Anwendung zur Berechnung des Roboterverhaltens und dessen graphischer Darstellung richtet beim Programmstart einen Socket

Server ein, der auf eine Verbindungsanfrage eines RCX Emulators wartet. Diese Verbindung dient dem Austausch von Motorzuständen, Sensorwerten und Befehlen. In jedem Simulationsschritt werden die aktuellen Motorzustände vom Emulator empfangen und die ermittelten Sensorwerte zurückgesendet. Auch Anweisungen wie »Neues Robotermodell laden« werden vom Emulator aus per Socket Verbindung an den Server geschickt.

Zu diesem Zweck wird eine Klasse *ClientSocket* bereitgestellt, die sowohl dafür zuständig ist, eingehende Nachrichten zu empfangen und an die Simulation weiterzuleiten, als auch die Sensordaten zu versenden.

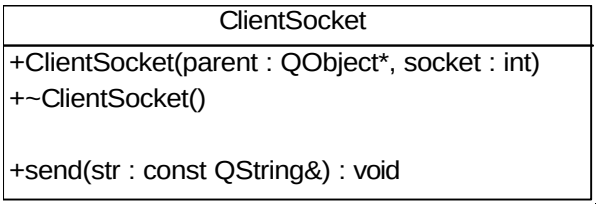

Abbildung 26: Klassenentwurf *ClientSocket*

Die Klasse *ClientSocket* wird von der Klasse *SimpleServer* erzeugt, die von der Klasse *QServerSocket* abgeleitet ist. Dadurch erhält die Klasse ClientSocket ohne die Benutzung von spezieller Systemprogrammierung die Funktionalität, mit Socket-Verbindungen arbeiten zu können.

6.6 Klassendiagramm

Der Gesamtentwurf der Anwendung zur Berechnung des Roboterverhaltens und dessen graphischer Ausgabe lässt sich am besten in Form eines Klassendiagramms darstellen:

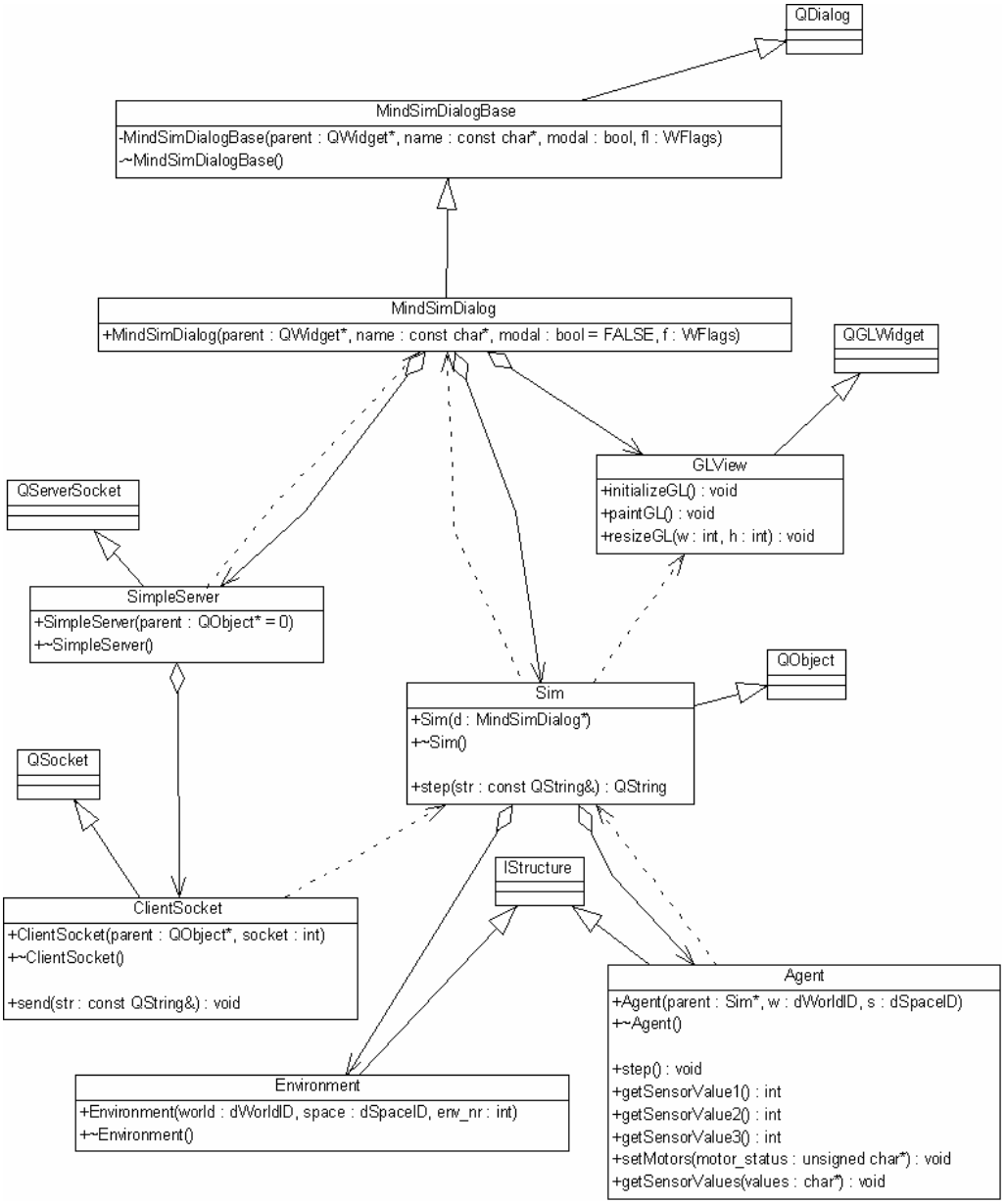

Abbildung 27: Klassendiagramm zum Simulationsserver

7 Das Produkt MindSim

Das vorliegende Produkt *MindSim*[40] ist ein voll funktionsfähiger Mindstorms™ Roboter Simulator, dessen Benutzung leicht zu erlernen ist. Eine Stunde Zeit reicht für Menschen mit Computererfahrung aus, um das Produkt zu installieren und erste kleine Roboterprogramme auszuprobieren.

Der Simulator besteht in dieser Version aus zwei Anwendungen, die getrennt gestartet werden:

- Der »MindSim Viewer« stellt das Roboterverhalten in einem frei skalierbaren Fenster graphisch dar.
- Der »MindSim Emulator« repräsentiert einen RCX mit den vier Bedienungsknöpfen. Zusätzlich enthält das Fenster dieser Anwendung die Menüsteuerung.

Produkteinsatz und Zielgruppe

Das Produkt ist für den Einsatz in folgenden Bereichen entwickelt worden:

- Heimbereich – als Ersatz für einen nicht vorhandenen LEGO® Mindstorms™ Baukasten.
- Schulungsbereich – zur Überbrückung von Hardware-Engpässen (Stehen z.B. für zehn Gruppen mit jeweils zwei oder drei Personen nur fünf Baukästen zur Verfügung, so können bei ausreichender PC-Versorgung einige Gruppen mit der Simulation arbeiten, bis ein realer Roboter verfügbar ist).

---

[40] Die Produktbezeichnung *MindSim* steht für »Mindstorms™ Simulator« und ist als reiner Arbeitstitel zu verstehen. Aufgrund von rechtlichen Bedenken bezüglich einer gleichnamigen Firma in den USA wird das Produkt nicht unter diesem Namen vetrieben werden.

- Laborbereich – zur Durchführung komplexer Experimente mit großen Umgebungen, die real nicht zur Verfügung stehen (z.B. ein Riesenlabyrinth).

Zu der Zielgruppe gehören alle Personen, die Interesse an LEGO® Mindstorms Robotern haben und auch ohne einen eigenen Baukasten Experimente damit durchführen wollen. Zur Hauptzielgruppe gehören in Zusammenhang mit dem Projekt Roberta zunächst KursleiterInnen, die sich ohne Verfügbarkeit eines Baukastens auf einen Roboterkurs vorbereiten möchten, und eventuell auch KursteilnehmerInnen bei Engpässen mit den realen Robotermodellen. Des weiteren gehören dazu Personen, die komplexe Experimente mit (räumlich) großen Umgebungen ohne viel Aufwand testen wollen.

7.1 Betriebssystemvoraussetzungen

Die vorliegende Version des Produkts ist für eine Ausführung auf Windows™ Systemen erstellt worden. Die Software kann unter Windows™ 95/98/2000/XP eingesetzt werden.

Die Taktfrequenz des eingesetzten Rechnersystems muss mindestens 600 MHz betragen, damit eine flüssige Bewegungsdarstellung des Roboterverhaltens gewährleistet ist. Des weiteren wird eine OpenGL-fähige Graphikkarte benötigt (zu OpenGL siehe Kapitel 4.5).

Die Bildschirmauflösung sollte bei mindestens 1024*768 Bildpunkten liegen, da die graphische Auflösung des Darstellungsfensters bei Programmstart 800x600 Bildpunkte beträgt und zusätzlich die Bedienungskonsole des virtuellen RCX dargestellt werden muss.

7.2 Installationsanleitung

Sämtliche, für die Installation benötigten Dateien, befinden sich auf der beiliegenden CD-ROM im Verzeichnis »Installation«. Die Beschreibung des Installationsvorgangs erfolgt hier beispielhaft für Windows XP™ mit »klassischem« Design. Der Vorgang ist aber bei anderen Windows™ Versionen und Designs sehr ähnlich. Es wird

vorausgesetzt, dass allgemeine Kenntnisse bezüglich des Kopierens von Dateien auf dem jeweils eingesetzten System vorhanden sind.

Die Installation des Simulators erfordert mehrere Schritte:

- Installation der Java™ Laufzeitumgebung: Java Runtime Environment (JRE)
- Installation des LEGO® Mindstorms™ Software Developer Kit (SDK) – (optional)
- Kopieren der für den MindSim Viewer benötigten Dateien
- Kopieren der für den MindSim Emulator benötigten Dateien
- Anlegen von Verknüpfungen

Legen Sie bitte die CD-ROM in das Laufwerk und wählen Sie auf dem jeweiligen Betriebssystem Ihren »Arbeitsplatz« aus. Wählen Sie dort das CD-Laufwerk mit der Bezeichnung »MindSim« aus.

Wechseln Sie nun in das Verzeichnis »Installation«. Es hängt von den Einstellungen Ihres Betriebssystems ab, in welcher Form Ihnen die Dateien im weiteren präsentiert werden. Je nachdem werden Endungen von Dateinamen wie z.B. ».exe« oder ».bat« angezeigt oder nicht. In dieser Installationsanleitung werden die Dateinamen daher immer mit Endung und Dateityp in Klammern angegeben. Der Dateityp wird unter Windows™ standardmäßig angezeigt.

Installation der Java™ Laufzeitumgebung

Falls auf Ihrem System noch keine aktuelle Java™ Laufzeitumgebung eingerichtet ist, muss diese zunächst installiert werden. Die Verwendung der Sun Java™ JRE ab einer Version von 1.4 wird dringend empfohlen. Sollte schon eine entsprechende Version auf dem Zielsystem eingerichtet sein, kann dieser Abschnitt übersprungen werden.

Wenn Sie auf der CD-ROM das Verzeichnis »Installation« ausgewählt haben, können Sie die Installation der aktuellen Version 1.4.2 der

JRE mit einem Doppelklick auf die Datei *j2re-1_4_2_02.exe (Anwendung)* starten.

Nach kurzer Zeit erscheint ein Fenster, in dem Sie den Lizenzvereinbarungen von Sun zustimmen müssen. Klicken Sie hier bitte einmal in das Feld neben dem Text: »Ich akzeptiere die Bedingungen der Lizenzvereinbarung« und bestätigen Sie dies mit einem Klick auf die Schaltfläche mit der Bezeichnung »Weiter«.

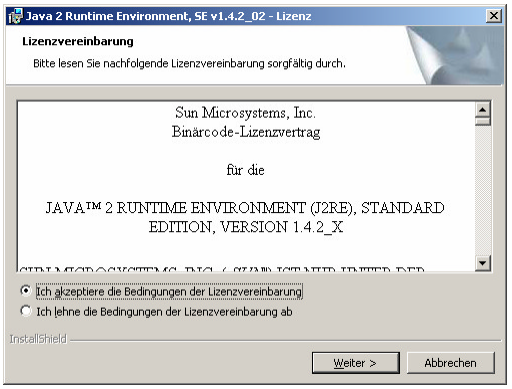

Abbildung 28: Lizenzvereinbarung der JRE

Daraufhin wird nach der Art der Installation gefragt. Sie können die Voreinstellung »Standard« übernehmen und einfach mit einem Klick auf »Weiter« bestätigen. Wenn Sie ein von Ihnen gewähltes Zielverzeichnis angeben möchten, wählen Sie an dieser Stelle »Angepasst« aus und folgen den weiteren Anweisungen.

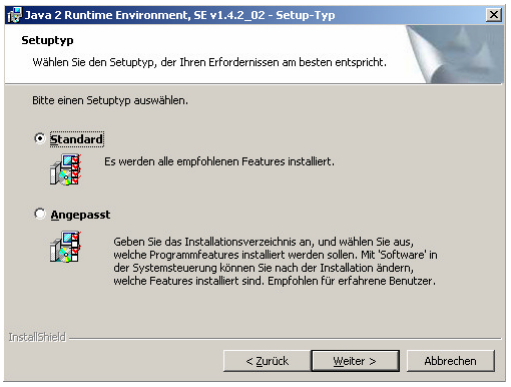

Abbildung 29: Auswahl des Installationstyps bei der JRE

Nach dem Kopieren der Dateien erscheint noch ein Fenster mit der Überschrift »InstallShield Wizard abgeschlossen«. Die Installation ist

hiermit beendet und Sie können mit einem Klick auf »Fertig stellen« den Vorgang abschließen.

Die Sun Java™ JRE ist nun auf Ihrem System eingerichtet. Sollte ein Neustart erforderlich sein, werden Sie hierzu aufgefordert.

Installation des Mindstorms™ Software Developer Kit (SDK)

Zum Import von mit der RIS Software erstellten Programmen, muss das von LEGO® frei erhältliche Software Developer Kit (SDK) installiert werden. Bei dessen Installation werden auf dem System Module eingerichtet, die zum Import der RIS Programme benötigt werden.

Wenn Sie auf der CD-ROM das Verzeichnis »Installation« ausgewählt haben, können Sie die Installation der aktuellen Version 2.5 des SDK mit einem Doppelklick auf die Datei *MindstormsSDK25.exe (Anwendung)* starten.

Nach kurzer Zeit erscheint ein Willkommensfenster, welches Sie durch anklicken von »Next« verlassen. Die Installationsroutine entpackt nun einige Dateien, was eine gewisse Zeit dauert. Das danach erscheinende Dialogfenster kann ebenfalls durch Anklicken von »Next« verlassen werden, worauf ein Fenster erscheint, in dem Sie den Lizenzvereinbarungen von LEGO® zustimmen müssen. Klicken Sie hier bitte einmal auf die Schaltfläche mit der Bezeichnung »Yes«.

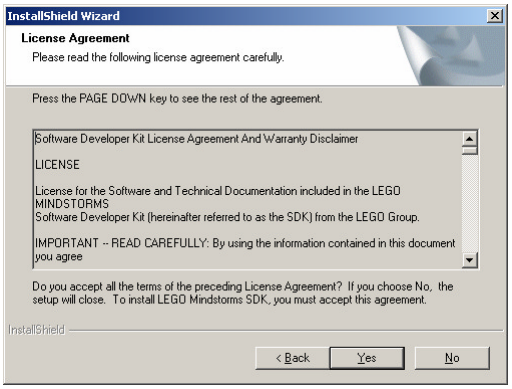

Abbildung 30: Lizenzvereinbarung des Mindstorms™ SDK

Wenn die Lizenzvereinbarung bestätigt worden ist, erscheint ein weiteres Fenster mit dem Titel »Information«. Verlassen Sie auch dieses Fenster mit einem Klick auf »Next«.

In dem darauf erscheinen Dialogfenster haben Sie die Möglichkeit, durch anklicken Schaltfläche »Browse« ein selbst gewähltes Installationsverzeichnis anzugeben. Sie können aber auch den Standardpfad übernehmen und die Installation einfach mit »Next« fortsetzen.

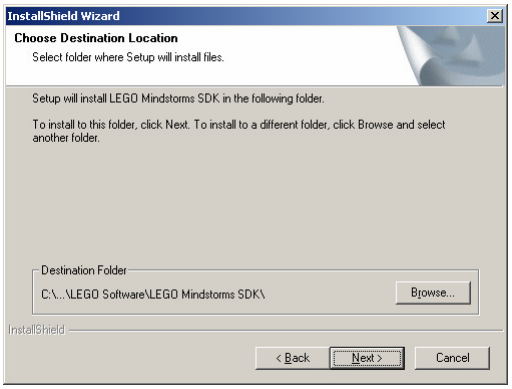

Abbildung 31: Installation SDK – Auswahl des Installationspfades

Im darauf folgenden Fenster werden Sie nach der Art der Installation gefragt. Sie sollten hier die Voreinstellung »Typical« übernehmen und mit einem Klick auf »Next« bestätigen.

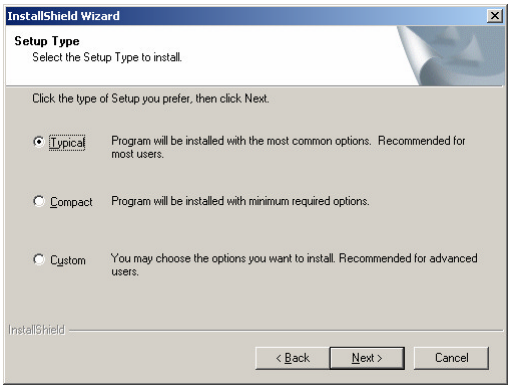

Abbildung 32: Auswahl des Installationstyps beim SDK

Schließlich erscheint noch ein Dialogfenster, in welchem Sie einen selbst gewählten Ordner im Windows™ Startmenü angeben können. Es wird empfohlen, die Standardeinstellung zu übernehmen und direkt mit einem Klick auf »Next« zu bestätigen.

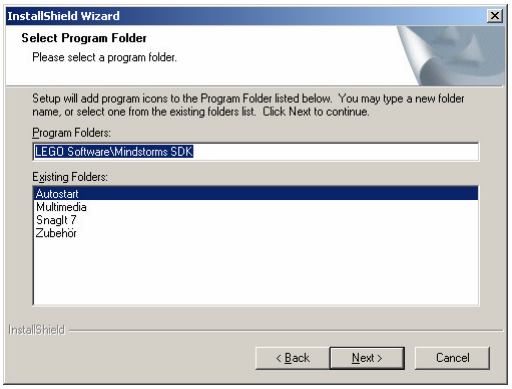

Abbildung 33: Auswahl des SDK Startmenü Ordners

Das nun erscheinende Fenster mit dem Titel »Start Copying Files« können Sie mit einem Klick auf »Next« verlassen, worauf das Kopieren der benötigten Dateien beginnt.

Zum Abschluss bekommen Sie noch eine Bestätigungsmeldung, die Sie mit einem Klick auf »Finish« beenden können. Sie haben jetzt Ihr System für den Import von RIS Programmen in den Simulator vorbereitet. Der Simulator greift automatisch auf die benötigten Module des SDK zu.

Kopieren der Dateien für den MindSim Viewer

Im Verzeichnis »Installation« auf der CD-ROM befindet sich ein Ordner mit der Bezeichnung »MindSimViewer«. Kopieren Sie diesen Ordner an einen Ort Ihrer Wahl. Am einfachsten ist es, den ganzen Ordner mit der Maus auf den Desktop zu ziehen. Hier kann dann auch jederzeit darauf zugegriffen werden.

Kopieren der Dateien für den MindSim Emulator

Die Dateien für den MindSim Emulator müssen ebenfalls auf den Zielrechner übertragen werden. Der zu kopierende Ordner nennt sich »MindSimEmulator« und befindet sich ebenfalls im Verzeichnis »Installation« auf der CD-ROM.

Anlegen von Verknüpfungen

Um eine komfortable Möglichkeit zu haben, den Simulator zu starten, bietet es sich an, Verknüpfungen für die beiden zugehörigen Anwendungen auf dem Desktop einzurichten.

Im Ordner des MindSim Viewers heißt die ausführbare Datei, die den Viewer startet, *MindSimViewer.exe (Anwendung)*.

Im Ordner des MindSim Emulators heißt die ausführbare Datei, die den Emulator startet, *MindSim.bat (Stapelverarbeitungsdatei)*.

Verknüpfungen können leicht erstellt werden, indem die auszuführende Datei mit der rechten Maustaste angeklickt wird. In dem erscheinenden Kontextmenü wählen Sie dann »Senden an -> Desktop (Verknüpfung erstellen)«. Daraufhin erscheint auf dem Desktop ein Symbol, mit dem die jeweilige Anwendung gestartet werden kann.

Wenn Sie für die beiden oben genannten ausführbaren Dateien Verknüpfungen eingerichtet haben, ist die Installation abgeschlossen und Sie können komfortabel mit dem Simulator arbeiten.

7.3 Benutzungsanleitung

Die vorliegende Version des Produkts ist als reine Testversion zu verstehen. Sie bietet nicht den für eine Veröffentlichung angedachten Komfort bezüglich Installation und Funktionsumfang, beinhaltet aber die grundlegenden Funktionen, die zum Testen notwendig sind.

Für die Benutzung des Simulators in dieser Version muss zuerst der MindSim Viewer und dann der MindSim Emulator gestartet werden. Der MindSim Viewer muss sichtbar sein, bevor der MindSim Emulator gestartet wird.

7.3.1 Der MindSim Viewer

Der Viewer wird entweder über die bei der Installation angelegte Verknüpfung oder durch Aufrufen der Datei *MindSimViewer.exe (Anwendung)* im Ordner »MindSimViewer« gestartet. Nach dem Start bietet sich folgendes Bild:

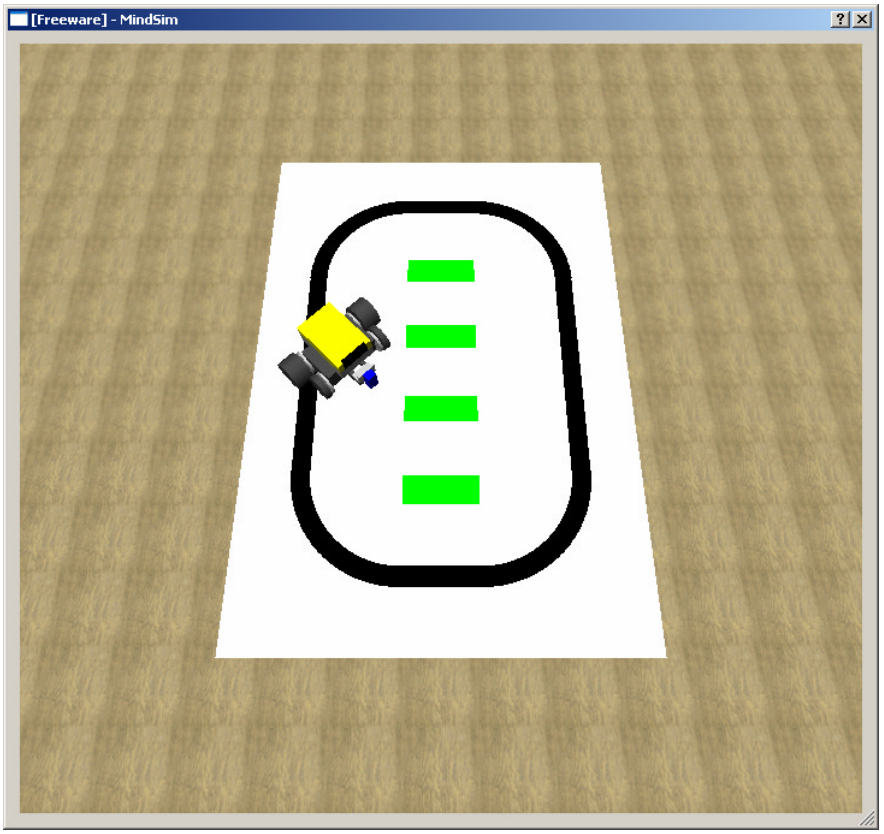

Abbildung 34: Startbildschirm des MindSim Viewers

Zu sehen ist zunächst eine Standardumgebung (die Nachbildung einer LEGO® Testunterlage) mit einem Roverbot, der mit einem Lichtsensor ausgestattet ist. Andere Umgebungen und Robotermodelle können vom Emulator aus in den Viewer geladen werden. Der Viewer selbst dient nur der Darstellung des Roboterverhaltens und der eventuellen Platzierung des Robotermodells.

Fenstergröße

Das Fenster ist in der Größe veränderbar, wobei der Fensterinhalt bei einer Änderung mitskaliert wird. Um die Größe des Fensters

anzupassen, bewegen Sie den Mauszeiger an dessen Rand. Es empfiehlt sich besonders die rechte, untere Ecke. Der Mauszeiger verändert sich dann in ein kleines Symbol mit zwei Pfeilen. Wenn Sie nun die linke Maustaste gedrückt halten und die Maus bewegen, passt sich die Fenstergröße automatisch an.

Kameraperspektive

Blickwinkel und »Standort« der fiktiven Kamera können frei gewählt werden. Führen Sie zu diesem Zweck zunächst den Mauszeiger über das Darstellungsfenster. Verschiedene Maustasten-Kombinationen in Verbindung mit einer Bewegung der Maus ermöglichen folgende Veränderungen der Kamera:

- bei gedrückter linker Maustaste: horizontale Veränderung des Standortes der Kamera
- bei gedrückter rechter Maustaste: vertikale und horizontale Veränderung der Ausrichtung der Kamera
- beide Maustasten gedrückt: vertikale Veränderung des Standortes der Kamera

Es ist schwierig, die Möglichkeiten genau zu beschreiben, die sich beim Einsatz der verschiedenen Maustasten-Kombinationen ergeben. In diesem Fall ist es sinnvoll, einfach ein wenig auszuprobieren, wie die Kamera mit der Maus gesteuert werden kann. Nach kurzer Gewöhnungszeit sollte es kein Problem darstellen, die Kamera per Maus in die gewünschte Position und Ausrichtung zu bringen, um den Bildausschnitt zu erhalten, der gerade erwünscht ist.

Position und Ausrichtung des Robotermodells

Das Robotermodell kann in seiner Umgebung frei bewegt und gedreht werden. Hierfür wird ebenfalls die Maus benutzt, wobei der Mauszeiger sich über dem Darstellungsfenster befinden muss. Um Änderungen an Position oder Ausrichtung vorzunehmen, muss zunächst die Alt-Taste auf der Tastatur gedrückt und festgehalten

werden. Ist diese Taste gedrückt, so lassen sich Änderungen am Robotermodell wie folgt vornehmen:

- bei gedrückter linker Maustaste: horizontale Veränderung der Position des Robotermdells
- bei gedrückter rechter Maustaste: Links- und Rechtsdrehung des Robotermodells
- beide Maustasten gedrückt: vertikale Veränderung der Position des Robotermodells

Auch in diesem Fall gilt: durch Ausprobieren werden die für erwünschte Veränderungen von Position und Ausrichtung des Robotermodells notwendigen Schritte schnell erlernt.

Beenden des Viewers

Zum Beenden des Viewers klicken Sie rechts oben im Fenster auf das kleine Kreuz.

7.3.2 Der MindSim Emulator

Der Emulator wird entweder über die bei der Installation angelegte Verknüpfung oder durch Aufrufen der Datei *MindSimEmulator.bat (Stapelverarbeitungsdatei)* im Ordner »MindSimViewer« gestartet. Wichtig ist, dass der MindSim Viewer schon läuft, bevor der Emulator gestartet wird. Nach dem Start bietet sich folgendes Bild:

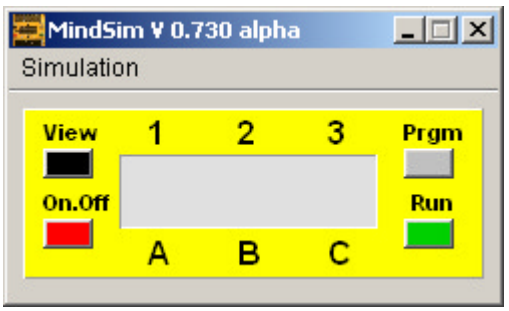

Abbildung 35: Benutzungsoberfläche des MindSim Emulators

Zu sehen ist die Nachbildung der Benutzungsoberfläche des RCX und eine Menüleiste mit dem Menü »Simulation«. Von dieser Anwendung aus wird die Simulation gesteuert.

Das Display

Das Display des RCX wird im Emulator nachgebildet. Es werden folgende Display-Funktionen unterstützt:

- Anzeige der seit dem Einschalten vergangenen Minuten (links neben dem kleinen Männchen).
- Anzeige des aktuellen Programmplatzes (rechts neben dem kleinen Männchen).
- Bei Programmausführung Statusanzeigen der Ausgänge A, B und C (kleine Pfeile, welche die Motorrichtung anzeigen).
- Bei Umschalten des Anzeigemodus mit der Taste View: Sensorwerte und Ausgangsleistungen.

Die Bedienungstasten

Der virtuelle RCX bietet, genauso wie sein physikalisches Vorbild, vier Bedienungstasten, die von der Funktionalität her genau die gleichen Auswirkungen haben, wie beim Original:

- Mit der Taste *On-Off* wird der RCX ein- und ausgeschaltet. Programme können nur im eingeschalteten Zustand geladen werden.
- Mit der Taste *Prgm* kann der aktuelle Programmspeicherplatz ausgewählt werden. Dieser wird im Display angezeigt. Jedes Drücken der Taste erhöht die Zahl des aktuellen Programmplatzes um eins. Bei Überschreiten der höchsten Programmplatz-Zahl von fünf, springt die Auswahl wieder auf den ersten Speicherplatz.

- Mit der Taste *Run* werden Programme gestartet und angehalten. Der Befehl bezieht sich auf den jeweils aktuellen Programmspeicherplatz.
- Durch (wiederholtes) Drücken der Taste *View* können nacheinander die verschiedenen Anzeigemodi auf dem Display umgeschaltet werden.

Die Menüpunkte

Über die Anwahl des Menüs »Simulation« lassen sich verschiedene Menüpunkte auswählen:

- Umgebung auswählen
- Robotermodell auswählen
- NQC Programm laden
- RIS Programm laden
- Beenden

Umgebung auswählen

Dieser Menüpunkt dient dazu, Umgebungen auszuwählen, in denen der Roboter sich bewegt. Es werden verschiedene, beispielhafte Umgebungen bereitgestellt. Bei Auswahl dieses Menüpunktes öffnet sich ein Dateiauswahlfenster. Die verfügbaren Umgebungen befinden sich im Unterordner »resources« des Ordners »MindSim-Emulator«. Sie haben die Dateiendung *.env (für Environment), wobei per Voreinstellung auch nur solche Dateien ausgewählt werden können.

In dieser Version gibt es drei Testumgebungen: Kreis, Raum und Testunterlage. Die Umgebung »Kreis« besteht aus einem schwarzen Kreis, der auf den Boden »gemalt« ist. Solch eine Umgebung eignet sich für Experimente mit Lichtsensoren. Ähnlich verhält es sich bei der Umgebung »Testunterlage« - auch diese ist für Lichtsensor-Experimente geeignet. Die Umgebung »Raum« stellt eine quadrati-

sche, durch »Stellwände« abgegrenzte Fläche dar. Diese eignet sich zum Experimentieren mit Berührungssensoren.

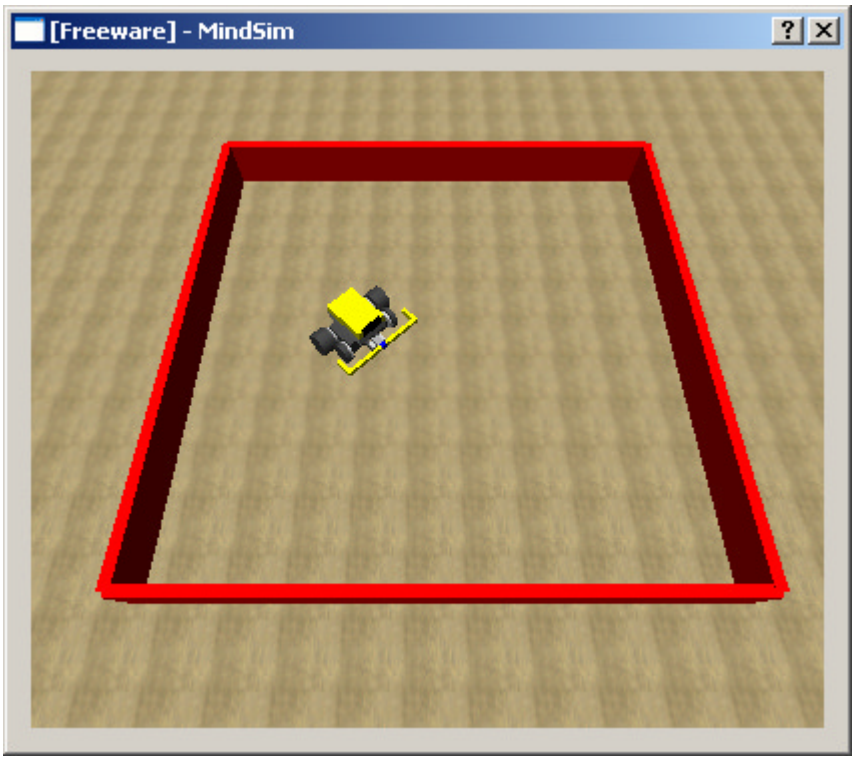

Abbildung 36: Testumgebung »Raum« für Berührungssensoren

Durch Auswahl der gewünschten Umgebung, wird der MindSim Viewer automatisch veranlasst, diese zu laden. Gleichzeitig wird für jede Umgebung ein zum Experimentieren geeignetes Robotermodell ausgewählt und geladen:

- Kreis: Roverbot mit einem Lichtsensor
- Testunterlage: Roverbot mit einem Lichtsensor
- Raum: Roverbot mit einem Berührungssensor

Robotermodell auswählen

Dieser Menüpunkt dient dazu, verschiedene Robotermodelle auszuwählen, die sich in ihrer Sensorausstattung unterscheiden. Es werden mehrere, beispielhafte Robotermodelle bereitgestellt. Bei Auswahl dieses Menüpunktes öffnet sich ein Dateiauswahlfenster. Die verfügbaren Roboter befinden sich im Unterordner »resources« des

Ordners »MindSimEmulator«. Sie haben die Dateiendung **.rob* (für Robot), wobei per Voreinstellung auch nur solche Dateien ausgewählt werden können.

In dieser Version gibt es vier Robotermodelle:

- Lichtsensor 1: ein Roverbot mit einem vorne angebrachten, nach unten gerichteten Lichtsensor
- Lichtsensor 2: ein Roverbot mit zwei vorne, nebeneinander angebrachten, nach unten gerichteten Lichtsensoren
- Touchsensor 1: ein Roverbot mit Stoßfänger vorne und einem auslösenden Berührungssensor
- Touchsensor 2: ein Roverbot mit Stoßfänger vorne und zwei Berührungssensoren für links und rechts

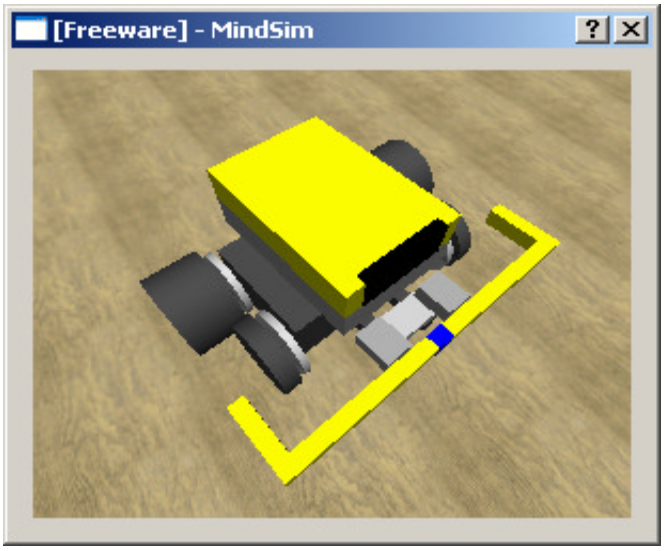

Abbildung 37: Ein Roboter mit zwei Berührungssensoren

Durch Auswahl des gewünschten Robotermodells, wird der MindSim Viewer automatisch veranlasst, dieses zu laden. Das vorher in der Simulationsumgebung befindliche Robotermodell wird entfernt.

NQC Programme laden

Durch Auswahl dieses Menüpunktes kann ein in NQC geschriebenes Roboterprogramm in den Emulator geladen werden. Dieses Pro-

gramm muss in Dateiform vorliegen. Typischerweise haben NQC Programme die Dateiendung **.nqc* – es können aber auch so genannte NQC-Listings mit der Endung **.lst* in den Emulator geladen werden. Zu beachten ist, dass der RCX eingeschaltet sein muss, um Programme laden zu können. Wird eine gültige, ausführbare Datei ausgewählt, so wird das in ihr enthaltene Programm auf den jeweils aktuellen Programmspeicherplatz des RCX geladen.

Im Unterordner »programs« des MindSim Emulators befinden sich einige NQC Beispielprogramme, die zum Testen verwendet werden können:

- *eingesperrt.nqc*: dieses Programm ist für den Roverbot mit einem Lichtsensor geschrieben. Es lässt den Roboter so umher fahren, dass er nicht über eine dunkle, am Boden befindliche Linie läuft. Bei Verwendung der Testumgebung »Kreis« hat dies z.B. den Effekt, dass der Roboter niemals aus dem Kreis läuft.

- *eingesperrt_raum.nqc* und *eingesperrt_raum2.nqc*: Diese Programme sind für den Roverbot mit Stoßfänger geschrieben. Sie veranlassen den Roboter dazu, umher zu fahren und bei Berührung einer Wand, zurückzusetzen und zu drehen. Die beiden Programme sind jeweils für das Modell mit einem und mit zwei Berührungssensoren gedacht.

- *linienfolger.nqc* und *linienfolger2.nqc*: Diese beiden Programme sind für den Roverbot mit Lichtsensor(en) geschrieben. Bei Ausführung fährt der Roboter an einer, am Boden befindlichen dunklen Linie entlang. Die beiden Programme sind jeweils für das Modell mit einem und mit zwei Lichtsensoren gedacht.

RIS Programme laden

Durch Auswahl dieses Menüpunktes kann ein mit RIS geschriebenes Roboterprogramm in den Emulator geladen werden. Dieses Programm muss in Dateiform vorliegen. Typischerweise haben RIS Programme die Dateiendung **.lsc* – es können aber auch RIS-Listings mit den Endungen **.prg* und **.rcx2* in den Emulator geladen werden. Zu beachten ist, dass der RCX eingeschaltet sein muss, um Programme laden zu können. Wird eine gültige, ausführbare Datei ausgewählt, so wird das in ihr enthaltene Programm auf den jeweils aktuellen Programmspeicherplatz des RCX geladen.

Im Unterordner »programs« des MindSim Emulators befinden sich einige RIS Beispielprogramme:

- *eingesperrt.lsc*, *eingesperrt_raum.lsc*, *eingesperrt_raum2.lsc, linienfolger.lsc* und *linienfolger2.lsc*: Diese Programme sind von der Ausführung her mit den gleichnamigen NQC Beispielprogrammen identisch. Eine Beschreibung findet sich im vorangegangenen Abschnitt.
- *musik.lsc*: Dieses Programm lässt den Roboter eine kurze Melodie spielen.
- *streifen-zaehlen.lsc*: Bei Ausführung dieses Programms fährt der Roboter fünf Sekunden lang vorwärts und zählt dabei dunkle Streifen oder Blöcke, die er mit einem nach unten gerichteten Lichtsensor erkennt. Danach gibt er die gleiche Anzahl von Piepstönen aus.

Beenden

Durch Auswahl dieses Menüpunktes wird der MindSim Emulator beendet. Der MindSim Viewer muss getrennt beendet werden.

8 Resümee und Ausblick

Ziel der vorliegenden Diplomarbeit war die Entwicklung eines Softwareprodukts. Zum Abschluss der Arbeit liegt eine voll funktionsfähige, zu Testzwecken benutzbare, Simulationsanwendung vor. Zur Implementation der Anwendung sind über 6000 Zeilen Quelltext in Java™ und über 2000 Zeilen Quelltext in C++ erstellt worden. Des weiteren ist die aus über 1500 Zeilen Quelltext bestehende ODL analysiert und an vielen Stellen verbessert und erweitert worden.

Die besondere Herausforderung bei der Produktentwicklung bestand in dem Entwurf einer plattformunabhängigen Implementation. Dieses Ziel ist durch die Verwendung der in der Arbeit genannten Techniken erreicht worden.

Das Produkt läuft in einer ersten Version unter Windows™ Systemen stabil und sicher. Sämtliche im Pflichtenheft genannten Tests lassen sich mit dem vorliegenden Produkt durchführen. Auch die Kriterien bezüglich des Echtzeitverhaltens und der Ausführbarkeit auf handelsüblichen PCs werden eingehalten.

Der Aufwand zur Entwicklung eines modularen Softwareprodukts mit diesem Umfang übersteigt schnell das im Rahmen einer Diplomarbeit machbare. Nur durch den intensiven Einsatz von Hilfsmitteln wie ODE und ODL konnte eine in sich abgeschlossene Produktentwicklung realisiert werden.

Das Produkt MindSim ist in der jetzigen Form jedoch nicht »fertig«. Es existieren viele Erweiterungsmöglichkeiten für den Mindstorms™ Simulator, die teilweise im Pflichtenheft festgehalten worden sind. Eine der interessantesten möglichen Erweiterungen ist die Unterstützung mehrerer Robotermodelle in einer Umgebung, die untereinander kommunizieren können. Abbildung 38 zeigt ein solches Szenario.

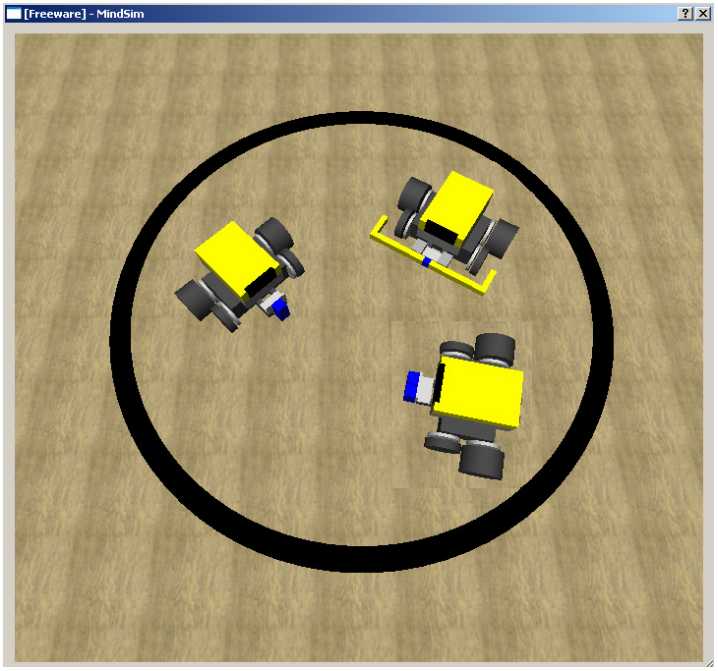

Abbildung 38: Ausblick - mehrere Robotermodelle in einer Umgebung

Denkbar sind im Zusammenhang mit der Mindstorms™ Simulation viele andere mögliche Erweiterungen. Vom Ausbau der virtuellen Maschine bis hin zu einer kompletten Hardware-Emulation des RCX ist hier alles machbar.

Die Weiterentwicklung des Produkts wird zunächst darauf basieren, das Produkt noch anwendungsfreundlicher zu gestalten. Ebenso werden noch nicht vorhandene Bearbeitungsroutinen für bestimmte Bytecode-Befehle (z.B. zur Erzeugung und Abfrage von Infrarot-nachrichten) nach und nach ergänzt.

Zum Schluss möchte ich mich bei den Mitarbeitern des Projekts Roberta für die gute Unterstützung während der gesamten Zeit der Arbeit bedanken. Insbesondere gilt der Dank meiner Kollegin Ulrike Petersen und meiner Betreuerin Monika Müllerburg, die mir beide viele gute Ratschläge gegeben haben. Bedanken möchte ich mich außerdem noch bei meiner betreuenden Professorin Prof. Dr. Heide Faeskorn-Woyke, die mir einen reibungslosen Ablauf der Arbeit ermöglicht hat.

Literaturverzeichnis

Baum, Dave: Dave Baums LEGO® Mindstorms™ Roboter – Der Profi-Guide, 1. Nachdruck, Bonn: Galileo Press, 2001.

Gorriz, C.M.; Medina, C.: Engaging Girls with Computers through Software Games, in: Communications of the ACM, January 2000, Vol. 43, No. 1, S. 42-49.

Flintrop, Björn: Programmieren von LEGO® Mindstorms™ Robotern mit brickOS, Projektarbeit Studium allg. Informatik, FH-Köln u. Fraunhofer Institut AIS, 3/2003.

Knudsen, Jonathan B.; Noga, Markus L.: Das Inoffizielle Handbuch für LEGO® Mindstorms™ Roboter, 1. Auflage, Köln: O'Reilly, 2000.

Knudsen, NZZ Folio 12/2000, »Papa spielen, Papa glücklich« <http://www-x.nzz.ch/folio/archiv/2000/12/articles/knudsen.html> (22.1.2004).

LEGO® Mindstorms™, RCX 2.0 Firmware, Command Overview, ELEC, 2000

Liebl, Franz: Simulation – Problemorientierte Einführung, 1. Auflage, München: Oldenburg, 1992.

Müllerburg, Monika et al., 2/2003, »Roberta – Informationen zum Projekt« <http://www.ais.fraunhofer.de/ROCK/roberta/documents/Informationen_ueber_Roberta.pdf> (26.1.2004).

Müllerburg, Monika et al., 5/2003, »Roberta – Projektblatt« <http://www.ais.fraunhofer.de/ROCK/roberta/documents/Roberta-PB.pdf> (27.1.2004).

Nagata, Joe: LEGO® Mindstorms™ Idea Book, 1. Auflage, San Francisco: No Starch Press, 2001.

Poole, T.G.; Szymankiewicz, J.Z.: Using Simulation to Solve Problems, London: 1977.

Reitman, J: Computer Simulation Applications – Discrete-Event Simulation for Synthesis and Analysis of Complex Systems, New York 1971.

Schoblick, Gabriele und Robert: LEGO® Mindstorms™ - Basics der Robotertechnik, 1. Auflage, Poing: Franzis, 2002.

Solomon, Susan L.: Building Modelers – Teaching the Art of Simulation, in: Interfaces, Vol. 10, #2, 1980, S. 65-72.

VDI-2860. VDI-Richtlinie 2860 - Handhabungsfunktionen, Handhabungseinrichtungen, Begriffe, Definitionen, Symbole, in: Entwurf, Jahrgang 1982, Berlin: Beuth, 1982

Wloka, Dieter W.: Robotersysteme 1 – Technische Grundlagen, 1. Auflage, Berlin-Heidelberg: Springer, 1992.

Wloka, Dieter W.: Robotersysteme 2 – Graphische Simulation, 1. Auflage, Berlin-Heidelberg: Springer, 1992.

Wloka, Dieter W.: Robotersysteme 3 – Wissenbasierte Simulation, 1. Auflage, Berlin-Heidelberg: Springer, 1992.

Anhang A – Tabelle der Bytecode Befehle

Die folgende Tabelle enthält eine Auflistung sämtlicher verfügbarer Bytecode Befehle mit dem jeweils zugehörigen Opcode, der Anzahl der Parameter und dem Bearbeitungsstatus. Beim Bearbeitungsstatus gelten folgende Abkürzungen: OK für »fertig implementiert«, D für ein nicht implementiertes, direktes Kommando und IR für Kommandos, welche die Infrarotschnittstelle ansprechen (ebenfalls nicht implementiert).

Befehl	Opcode	Parameter	Status
PBAliveOrNot	0x10	0	OK
MemMap	0x20	0	D
PBBattery	0x30	0	D
DeleteAllTasks	0x40	0	D
StopAllTasks	0x50	0	OK
PBTurnOff	0x60	0	OK
DeleteAllSubs	0x70	0	D
ClearSound	0x80	0	OK
ClearPBMessage	0x90	0	OK
ExitAccessControl	0xa0	0	OK
ExitEventCheck	0xb0	0	OK
MuteSound	0xd0	0	OK
UnmuteSound	0xe0	0	OK
ClearAllEvents	0x06	0	OK
EndOfSub	0xf6	0	OK
OnOffFloat	0x21	1	OK
PbTXPower	0x31	1	IR
PlaySystemSound	0x51	1	OK
DeleteTask	0x61	1	D
StartTask	0x71	1	OK
StopTask	0x81	1	OK
SelectProgram	0x91	1	OK
ClearTimer	0xa1	1	OK

PBPowerDownTime	0xb1	1	OK
DeleteSub	0xc1	1	D
ClearSensorValue	0xd1	1	OK
SetFwdSetRwdRewDir	0xe1	1	OK
Gosub	0x17	1	OK
SJump	0x27	1	OK
SCheckLoopCounter	0x37	1	OK
ConnectDisconnect	0x67	1	OK
SetNormSetInvAltDir	0x77	1	OK
IncCounter	0x97	1	OK
DecCounter	0xa7	1	OK
ClearCounter	0xb7	1	OK
SetPriority	0xd7	1	OK
InternMessage	0xf7	1	IR
PlayToneVar	0x02	2	OK
Poll	0x12	2	D
SetWatch	0x22	2	OK
SetSensorType	0x32	2	OK
SetSensorMode	0x42	2	OK
SetDataLog	0x52	2	OK
DataLogNext	0x62	2	OK
LJump	0x72	2	OK
SetLoopCounter	0x82	2	OK
LCheckLoopCounter	0x92	2	OK
SendPBMessage	0xb2	2	IR
SendUARTData	0xc2	2	IR
RemoteCommand	0xd2	2	D
SDecVarJumpLTZero	0xf2	2	OK
DirectEvent	0x03	3	OK
SetPower	0x13	3	OK
PlayTone	0x23	3	OK
SelectDisplay	0x33	3	OK
Wait	0x43	3	OK

UploadRam	0x63	3	D
EnterAccessControl	0x73	3	OK
SetEvent	0x93	3	OK
SetMaxPower	0xa3	3	OK
LDecVarJumpLTZero	0xf3	3	OK
CalibrateEvent	0x04	4	OK
SetVar	0x14	4	OK
SumVar	0x24	4	OK
SubVar	0x34	4	OK
DivVar	0x44	4	OK
MulVar	0x54	4	OK
SgnVar	0x64	4	OK
AbsVar	0x74	4	OK
AndVar	0x84	4	OK
OrVar	0x94	4	OK
Upload	0xa4	4	D
SEnterEventCheck	0xb4	4	OK
SetSourceValue	0x05	5	OK
UnlockPBrick	0x15	5	D
BeginOfTask	0x25	5	D
BeginOfSub	0x35	5	D
ContinueDL	0x45	5	D
GoIntoBootMode	0x65	5	D
BeginFirmwareDownload	0x75	5	D
SCheckDo	0x85	5	OK
LCheckDo	0x95	5	OK
Unlock Firmware	0xa5	5	D
LEnterEventCheck	0xb5	5	OK
ViewSourceValue	0xe5	5	OK

Tabelle 1: Bytecode-Kommandos und Implementationsstatus

Anhang B – Tabelle der Parameter Sources

Source	Name	Range	Comments
0	Variable	0-47	Variables are 16-bit signed values. Immediate commands can not access local variables. Value: -32768 - 32767
1	Timer	0-3	Timers are 15-bit unsigned (actually 16-bit signed with 0x7FFF-0 wrap-around) with 100 ms resolution. Value: 0-32767
2	Constant	-32768 - 32767	Different command accept different actual values
3	Motor status	0-2	The values are 8-bit wide fields, formatted as: bit 0-2: power setting bit 3-3: direction (1=fwd, 0=rwd) bit 4-5: remote command marker bit 6-6: brake/float (1=brake) bit 7-7: on/off (1=on, 0=off) when bit 7 is set (the motor is on), bit 6 is ignored
4	Random	1-32767	The source generates a random number between 0 and the actual given range. Writing to random 0 seeds the random number generator.
8	Program slot	0-4	Writing changes the selected program slot
9	Sensor value	0-2	The returned value is generated in the firmware based on sensor type and mode

10	Sensor type	0-2	The values are 8-bit numbers, meaning: 0: No sensor 1: Switch 2: Temperature 3: Reflection 4: Angle
11	Sensor mode	0-2	The values are 8-bit fields, with the bits meaning: 0x00: RawMode 0x20: BooleanMode 0x40: TransitionCntMode 0x60: PeriodCounterMode 0x80: PctFullScaleMode 0xA0: CelsiusMode 0xC0: FahrenheitMode 0xE0: AngleStepsMode
12	Sensor raw	0-2	The un-processed 10 bit A/D value, straight off the hardware. Value: 0 - 1023
13	Sensor bool	0-2	The raw value, with tresholds and hysteresis at 45/55%. Value: 0-1
14	Watch	0	The source is interpreted as "hours*60 + minutes" and cannot exceed 23 hrs and 59 min.
15	Message	0	The value is an 8-bit unsigned number received from another P-brick.
17	Global motor status	0-2	The values are 8-bit wide fields, formatted as mentioned in source 3.
21	Counter	0-2	The counters are identical to global variables 0-2. The difference is that they can be used to generate events based on thresholds. Value: -32768 - 32767
23	Task events	0-10	Contains a copy of the bit register of the event(s) that lead to event monitoring succeeding. In tasks

			and subroutines (mainly), the index 10 means "current task".
25	Event state	0-15	Tells wether the event is in the low, normal or high state depending on the set thresholds. The event state may also be undefined or calibrating. 0: Low 1: Normal 2: High 3: Undefined 4: Start calibrating 5: Calibrating in process
26	10 ms timer	0-3	Timers are 15-bit unsigned (actually 16-bit signed with 0x7FFF-0 wrap-around) with 10 ms resolution. These timers are derived from source 1 and only generated on demand. Value: 0-32767
27	Click counter	0-15	Contains the number of Clicks detected by firmware for the event sensor. Value: -32768 - 32767
28	Upper threshold	0-15	Used for event generation. The actual values should be in native units (source 0, 1, 9, 15 and 21). When specifying temperature, thresholds the threshold must be given as temperature multiplied by 10. Value: -32768 - 32767
29	Lower threshold	0-15	As above.
30	Hysteresis	0-15	As above. The value must be non-negative. Value: 0 - 32767

31	Duration	0-15	The value is measured in 10 ms units. Value: 5 - 32767 (50 ms - ca. 327 s)
33	UART setup	0-17	When writing to the source, only the lower 8 bit of the value are used.
34	Battery level	0	The value is the battery voltage level multiplied by 1000.
35	Firmware version	0	The value is the firmware version number multiplied by 100.
36	Indirect Variable	0-47	This source first looks up the indicated (index/pointer) variable and then uses the value of that variable to look up the final variable which is either read from or written to.

Tabelle 2: Tabelle der Parameter Sources

Anhang C – Inhalt der beigefügten CD-ROM

Auf der beigefügten CD-ROM befinden sich folgende Ordner und Dateien:

- der Ordner »Dokumentation« enthält die, mit Javadoc erstellte, Quelltextdokumentation zum MindSim Emulator. Sie wird durch einen Aufruf der Datei »index.html« geöffnet.
- der Ordner »Installation« enthält sämtliche für die Installation des Produkts notwendigen Dateien.
- der Ordner »Literatur« enthält Dokumente über die ODE und den RCX Bytecode.
- der Ordner »Quelltext« enthält die kompletten, für eine Kompilierung des Produkts, notwendigen Quellen.
- Die schriftliche Ausarbeitung der Arbeit in Form der Datei »Diplom.pdf«.
- Diese Inhaltsangabe in Form der Datei »readme.txt«.

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für diese Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, den 5.3.2004,

Björn Flintrop